CO-UNIT VI- Basic Processing Unit

(HARDWIRED CONTROL AND MICROPROGRAMMED CONTROL)

Text/Reference Books

The following sources are used for preparing these slides

1. Computer Organization **,** Carl Hamacher, Zvonko Vranesic, Safwat Zaky McGraw Hill Publ.

2. Computer Organization and Architecture**:** Designing for Performance**,** William Stallings , Prentice-Hall India,Publ.

3.Computer Architecture A Quantitative Approach ,John L Hennessy and David Patterson , Morgan Kaufman Publ.

4.Structured Computer Organization ,Andrew S. Tanenbaum , Prentice-Hall India Publ.

5. Computer Organization and Design, P. Paul Choudhury ,Prentice-Hall India,Publ.

Websites:

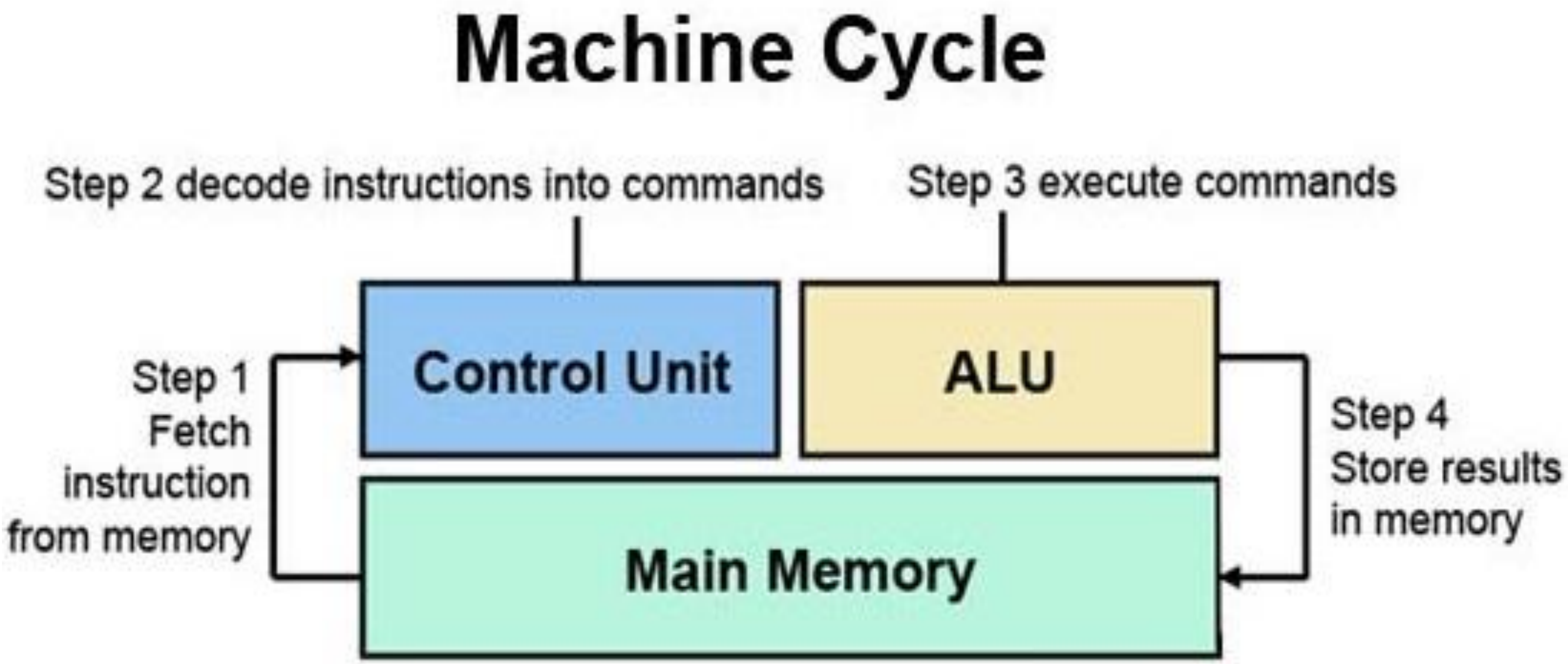
CONTENTS

CO-UNIT VI
BASIC PROCESSING UNIT

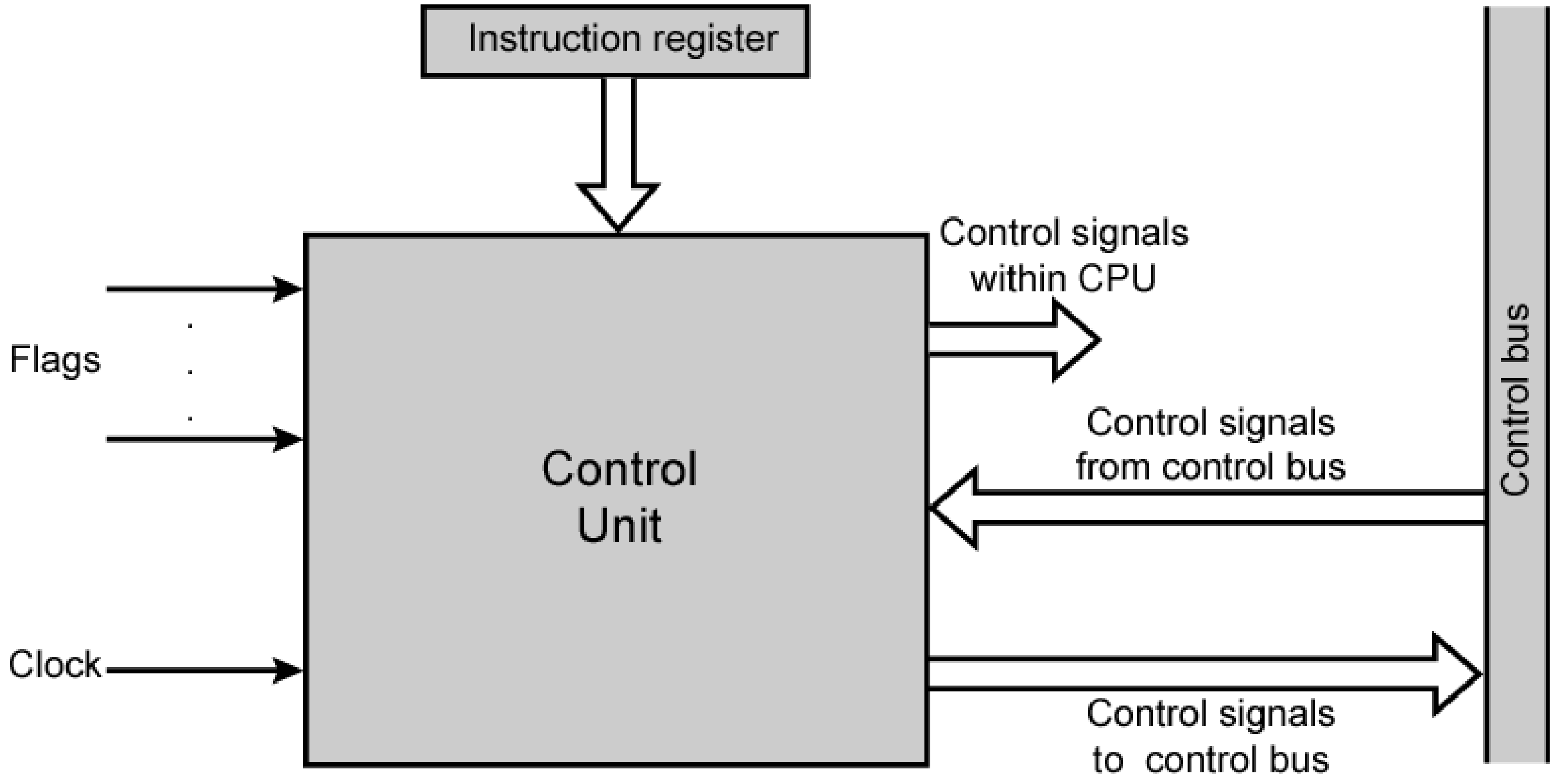
Dt.8.3.18

Updated on 13.3.18

Machine Cycle


Machine Cycle
Step 2 decode instructions into commands
Step 3 execute commands
Step 1 Fetch instruction from memory
Control Unit
ALU
Step 4 Store results in memory
Main Memory

Model of Control Unit


Instruction register
Control signals within CPU
Flags
Control Unit
Control signals from control bus
Control bus
Clock
Control signals to control bus

Functions of Control Unit using Control Signals

- Sequencing
  - CU causes the CPU to step through a series of micro-operations in proper sequence based on the program being executed.

  E.g. In order to carry out a task such as ADD, the control unit must generate a set of control signals in a predefined sequence governed by the HW structure of the processing section.
- Execution
  - CU causes each micro-operation to be performed
- Control Signals
  - External: inputs indicating the state of the system
  - Internal: logic required to perform the sequencing and execution functions

Fundamental Concepts contd..

- Inputs to control unit are:
  - Master clock
  - Status info from processing section
  - Command signals from external agent
- Outputs produced by control unit
  - Signals that drive the processing section and responses to an external envent (operation complete or abort) due to exceptions (overflow and underflow)
- Control unit undertakes the following responsibilities
  - Instruction interpretation:

    Read instr. , Recognize, Get operands and Route to appropriate functional units, necessary control signals issued
  - Instruction sequencing:

    Control unit determines the address of next instruction to be executed and loads to PC
- Processor fetches one instruction at a time and perform the operation specified.
- Instructions are fetched from successive memory locations until a branch or a jump instruction is encountered

Connection between CPU and Memory

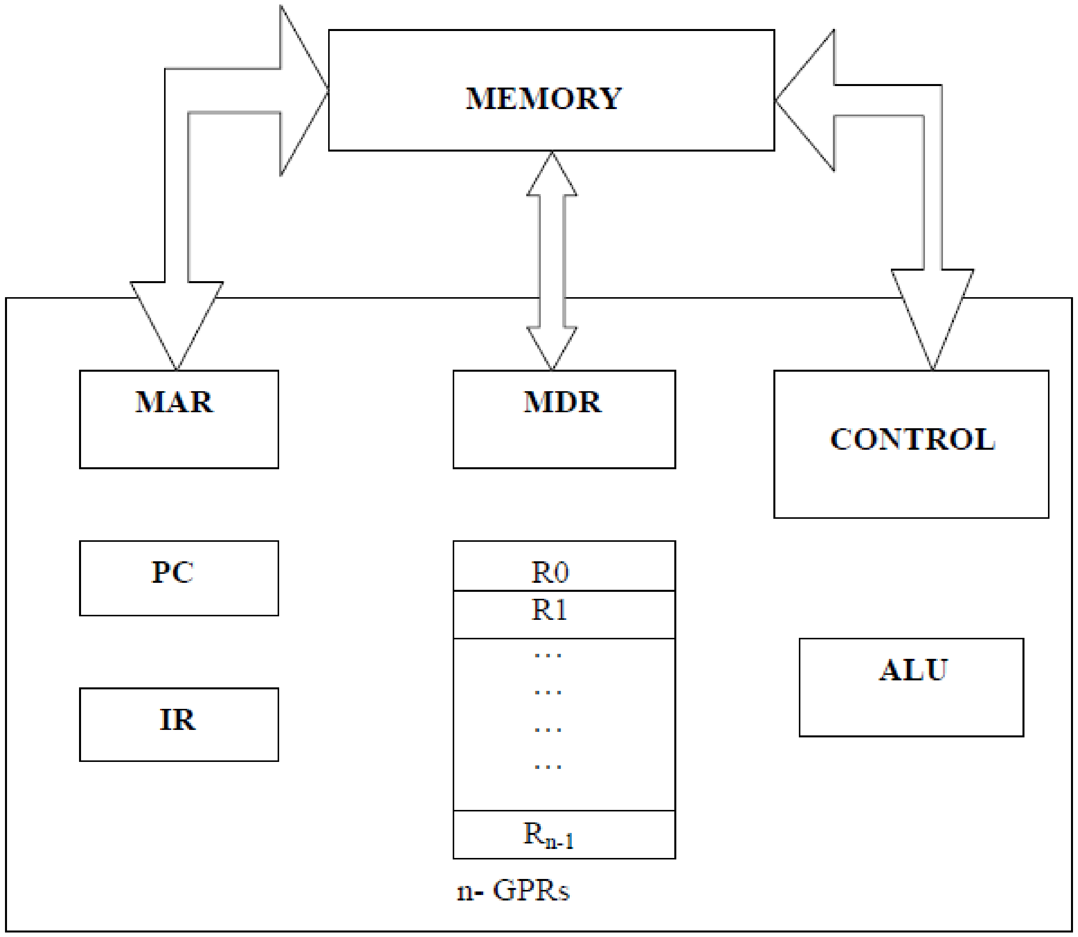


Fig b : Connections between the processor and the memory

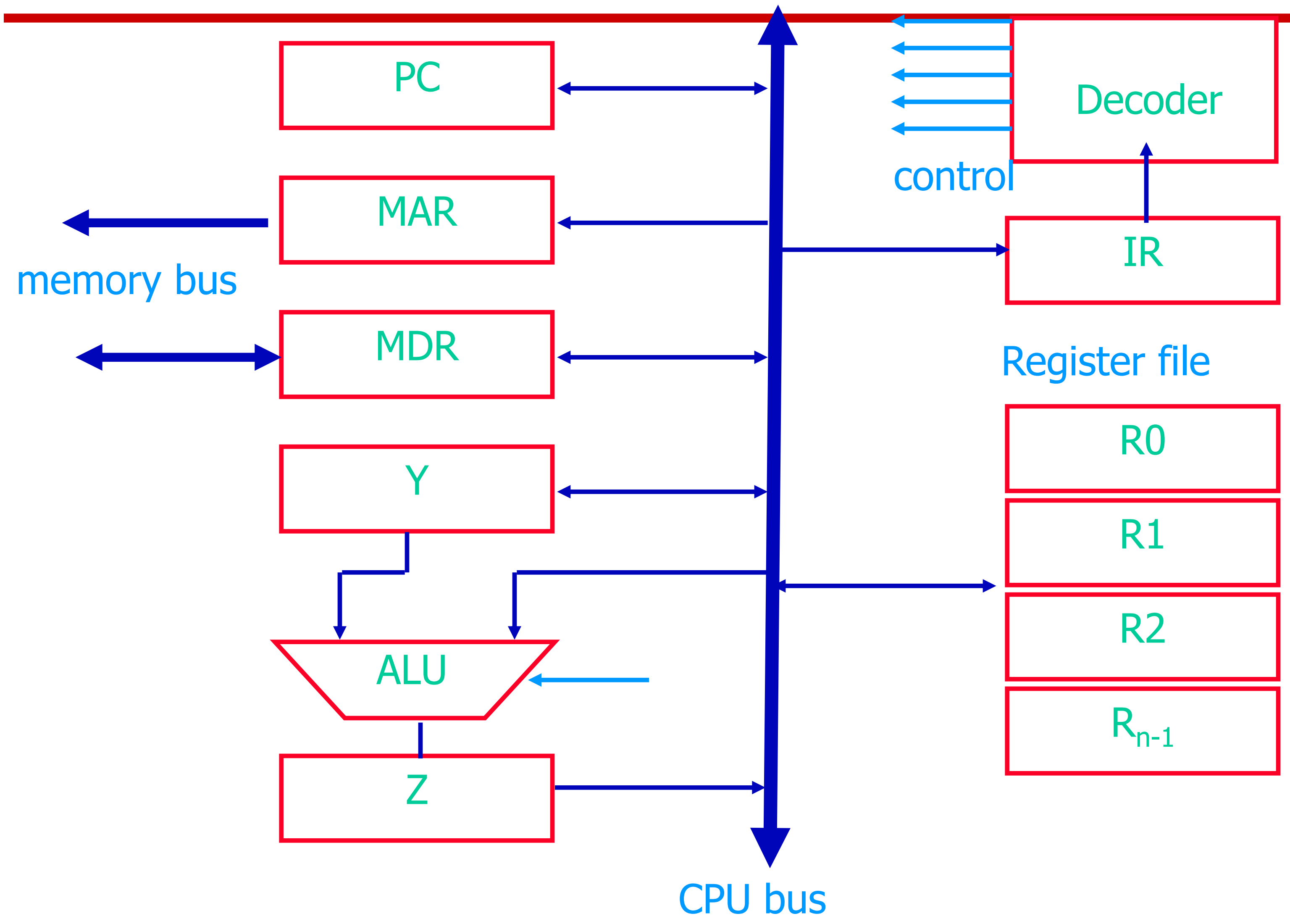
Fetch/execute cycle

- Step I:
  - Fetch the contents of the memory location pointed to by Program Counter (PC).
  - PC points to the memory location which has the instruction to be executed.
  - Load the contents of the memory location into Instruction Register (IR).
- Step II:
  - Increment the contents of the PC by 4 (assuming the memory is byte addressable and the word length is 32 bits).
- Step III:
  - Carry out the operation specified by the instructions in the IR.
- Steps I and II constitute the fetch phase, and are repeated as many times as necessary to fetch the complete instruction.
- Step III constitutes the execution phase.

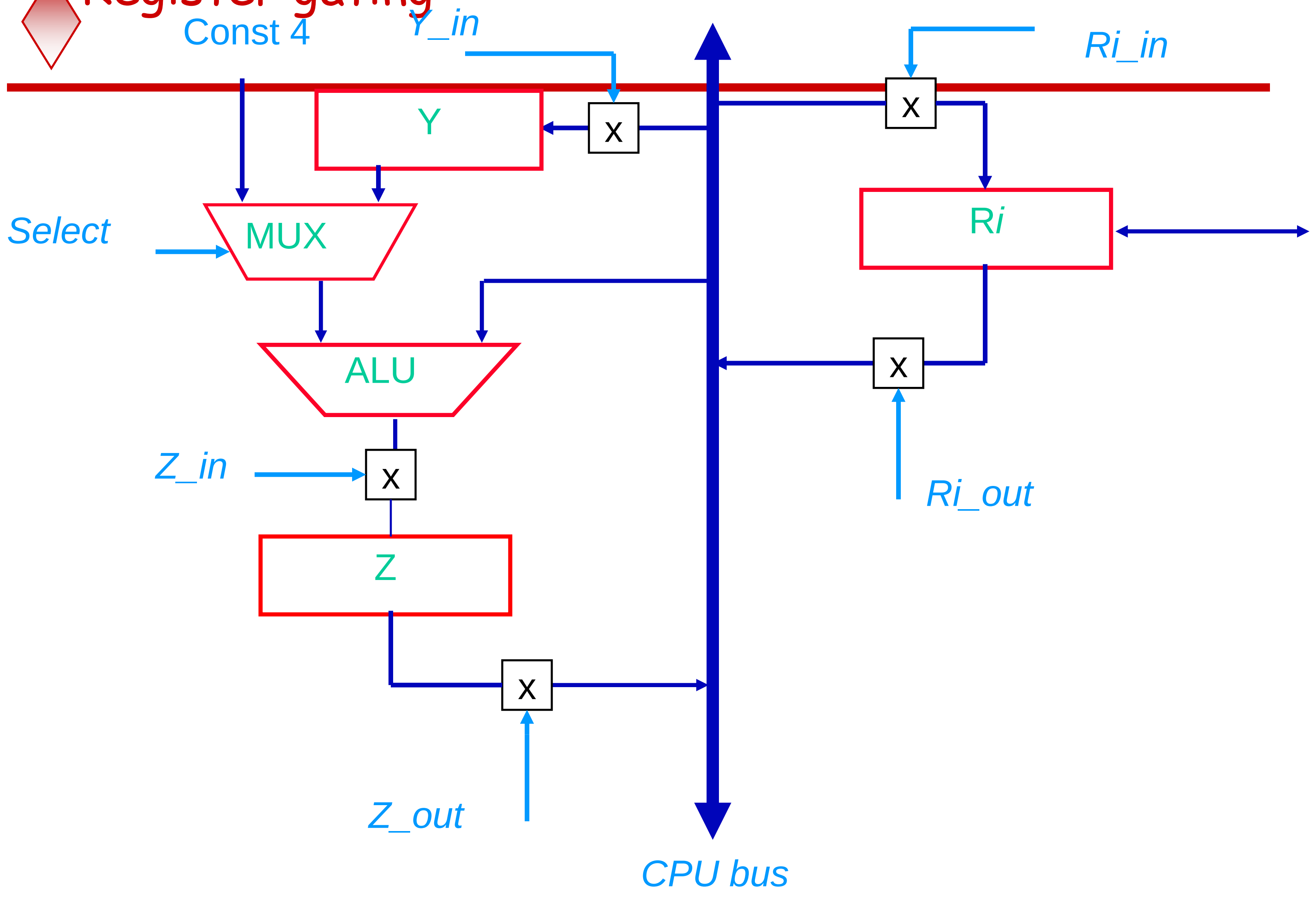
Internal organization of a processor

- Recall that a processor has several registers/building blocks:
  - Memory address register (MAR)
  - Memory data register (MDR)
  - Program Counter (PC)
  - Instruction Register (IR)
  - General purpose registers R0 - R(n-1)
  - Arithmetic and logic unit (ALU)
  - Control unit.
- How are these units organized and how do they communicate with each other?

Organization



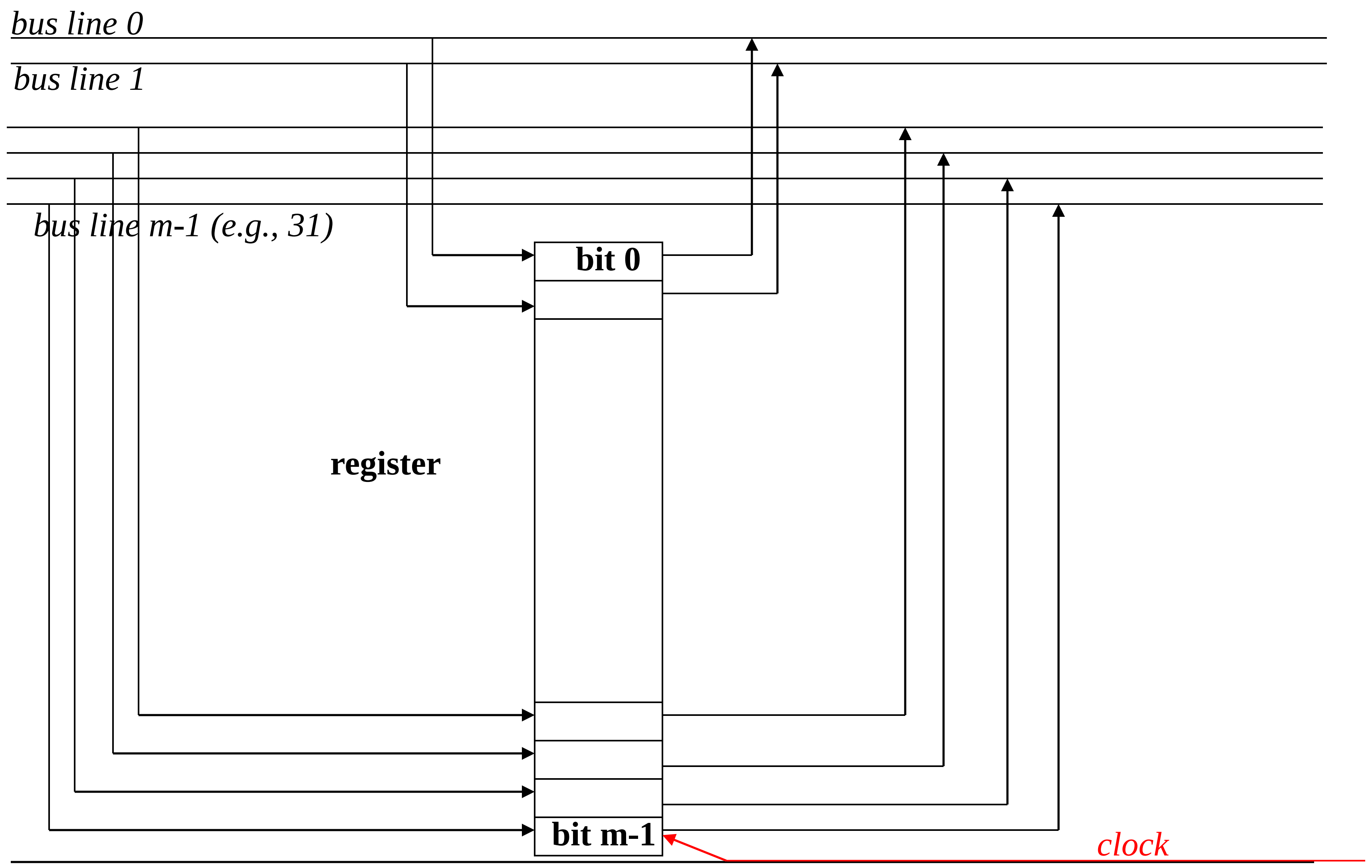
Register gating



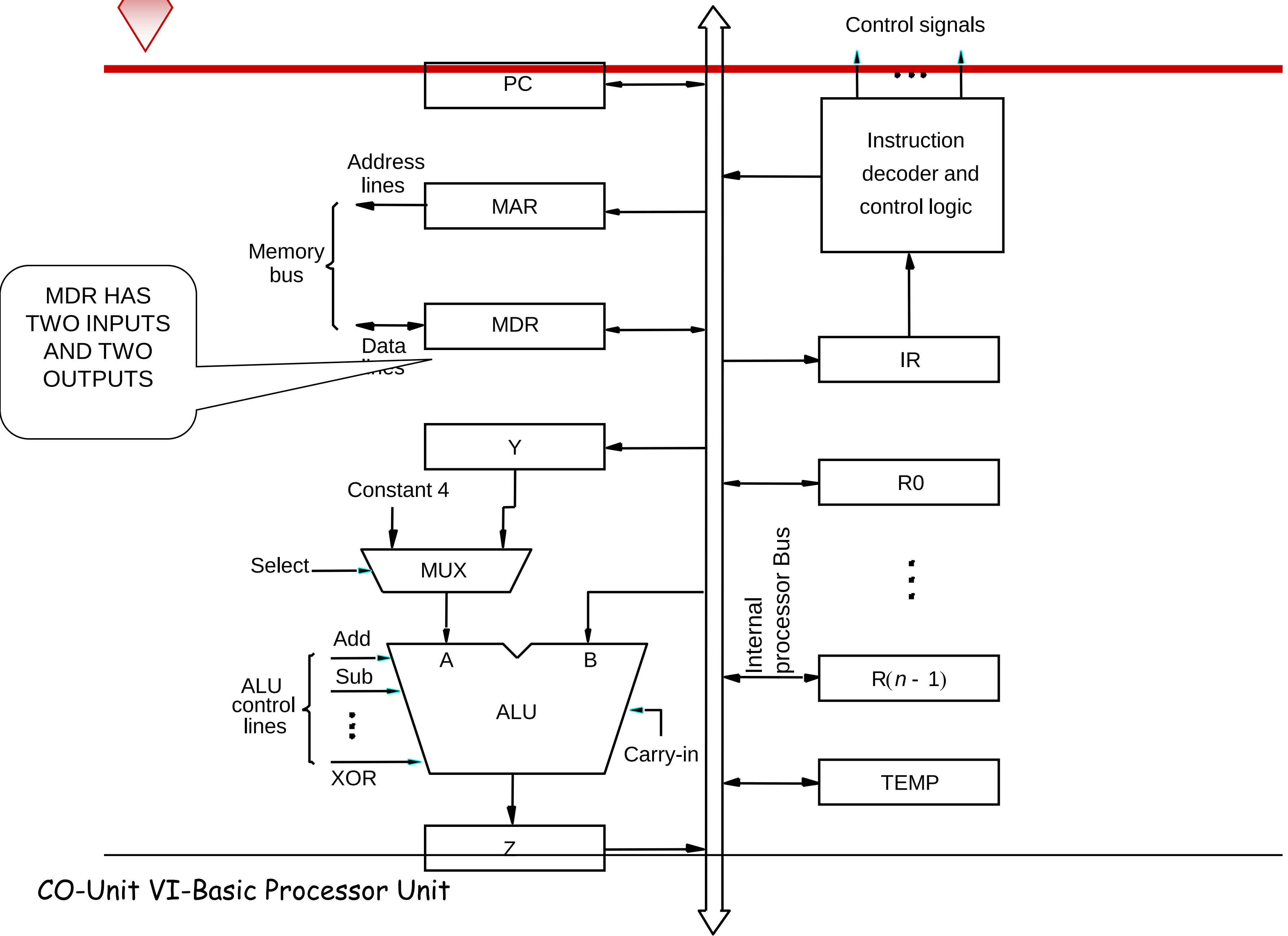
Input & Output Gating for one Register Bit

- A 2-input multiplexer is used to select the data applied to the input of an edge-triggered D flip-flop.
- When Riin =1, mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock.
  When Riin =0, mux feeds back the value currently stored in flip-flop.
- Q output of flip-flop is connected to bus via a tri-state gate.
  When Riout = 0, gate's output is in the high-impedance state.
  (This corresponds to the open- circuit state of a switch).
  When Riout =1, the gate drives the bus to 0 or 1, depending on the value of Q.

Registers and the bus



Internal organization of a processor



Single bus organization

- Single bus organization:
  - ALU, control unit and all the registers are connected via a single common bus (Called Internal Bus)
  - Bus is internal to the processor and should not be confused with the external bus that connects the processor to the memory and I/O devices.
- Data lines of the external memory bus are connected to the internal processor bus via MDR.
  - Register MDR has two inputs and two outputs.
  - Data may be loaded to (from) MDR from (to) internal processor bus or external memory bus.
- Address lines of the external memory bus are connected to the internal processor bus via MAR.
  - MAR receives input from the internal processor bus.
  - MAR provides output to external memory bus.

Single bus organization (contd..)

- Instruction decoder and control logic block, or control unit issues signals to control the operation of all units inside the processor and for interacting with the memory bus.
  - Control signals depend on the instruction loaded in the Instruction Register (IR)
- Outputs from the control logic block are connected to:
  - Control lines of the memory bus.
  - ALU, to determine which operation is to be performed.
  - Select input of the multiplexer MUX to select between Register Y and constant 4.
  - Control lines of the registers, to select the registers.

Single bus organization (contd..)

- Registers Y, Z, and TEMP:
  - Used by the processor for temporary storage <u>during</u> execution of some instructions.
  - Note that Registers R0 to R(n-1) are used to store data generated by one instruction for later use by another instruction.
  - Data is stored in R0 through R(n-1) <u>after</u> the execution of an instruction.
- Multiplexer MUX selects either the output of register Y or a constant 4, depending upon the control input Select.
  - Constant 4 is used to increment the value of the PC.
- B input of ALU is obtained directly from processor-bus.

  As instruction execution progresses, data are transferred from one register to another, often passing through ALU to perform arithmetic or logic operation.

Registers and the bus (contd..)

- A bus may be viewed as a collection of parallel wires.
- Buses have no memory:
  - They are just a collection of wires.
- When data is on the bus, all registers can "see" that data at their inputs.
- A register may place its contents onto the bus.

Registers and the bus (contd..)

- At any one time, only one register may output its contents to the bus:
  - Which register outputs its content to the bus is determined by the control signal issued by the control logic.
  - Control signal depends on the instruction loaded in the instruction register.
- Registers can load data from the bus:
  - Which registers load data from the bus is determined by the control signal issued by the control logic.
- Registers are clocked (sequential) entities (unlike ALU which is purely combinatorial).

The Processing Unit

1. Basic Processing Cycle
2. Types of Operations
   1. Register Transfer
   2. Fetch from Memory
   3. Store to Memory
   4. Arithmetic/Logic Ops.
   5. Execution of Complete Instruction
   6. Branching Ops.
3. Control Mechanisms

2. Types of Operations

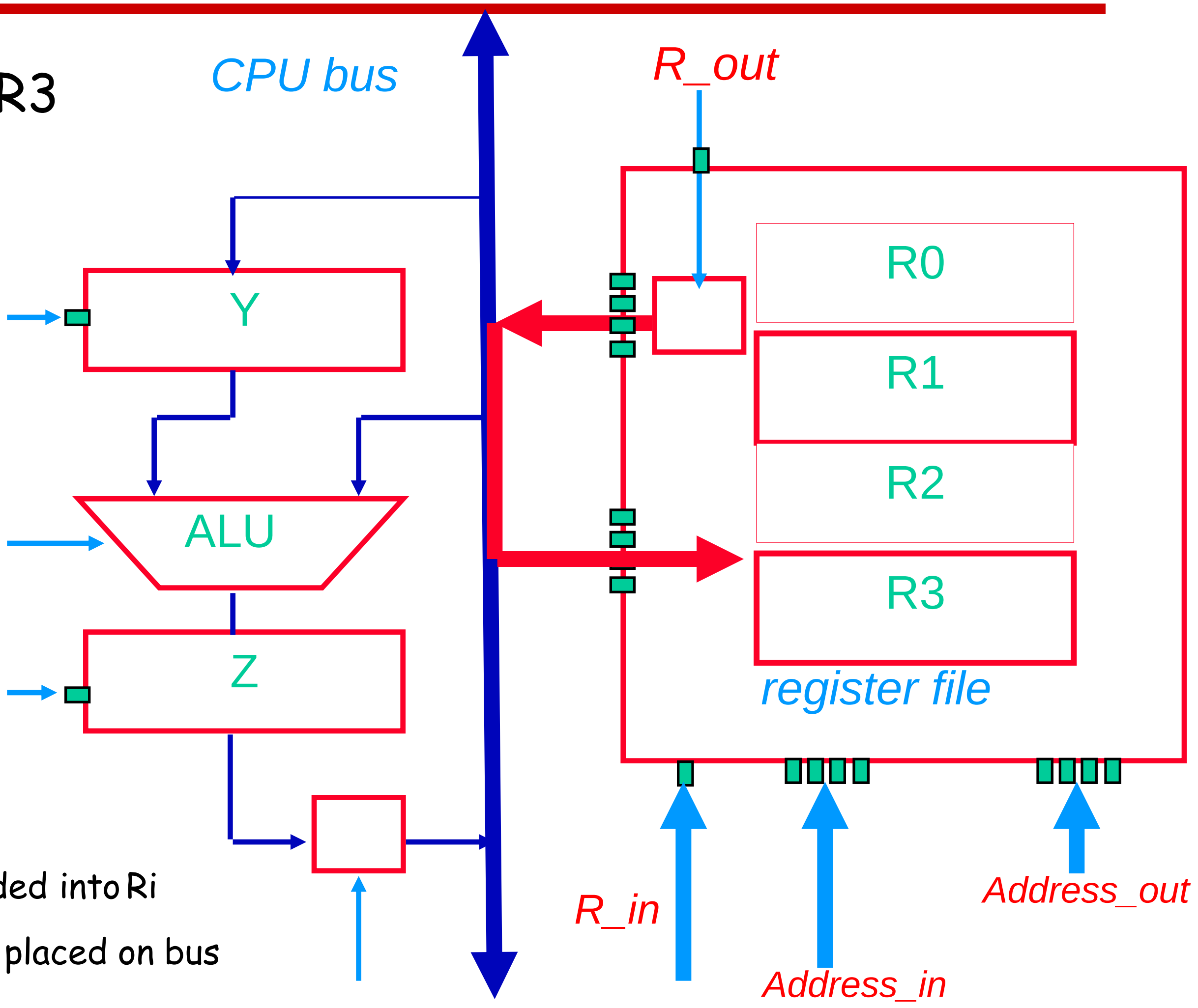
- Operation cycle includes:
  - Transfer data from register to register or to ALU
  - Fetch contents of memory location and put in one of the CPU registers
  - Store contents of CPU register in memory location
  - Perform arithmetic or logic operation

2.1. Register Transfers

Copy contents of R1 to R3

1. Address_out=R1
2. R_out
3. Address_in=R3
4. R_in

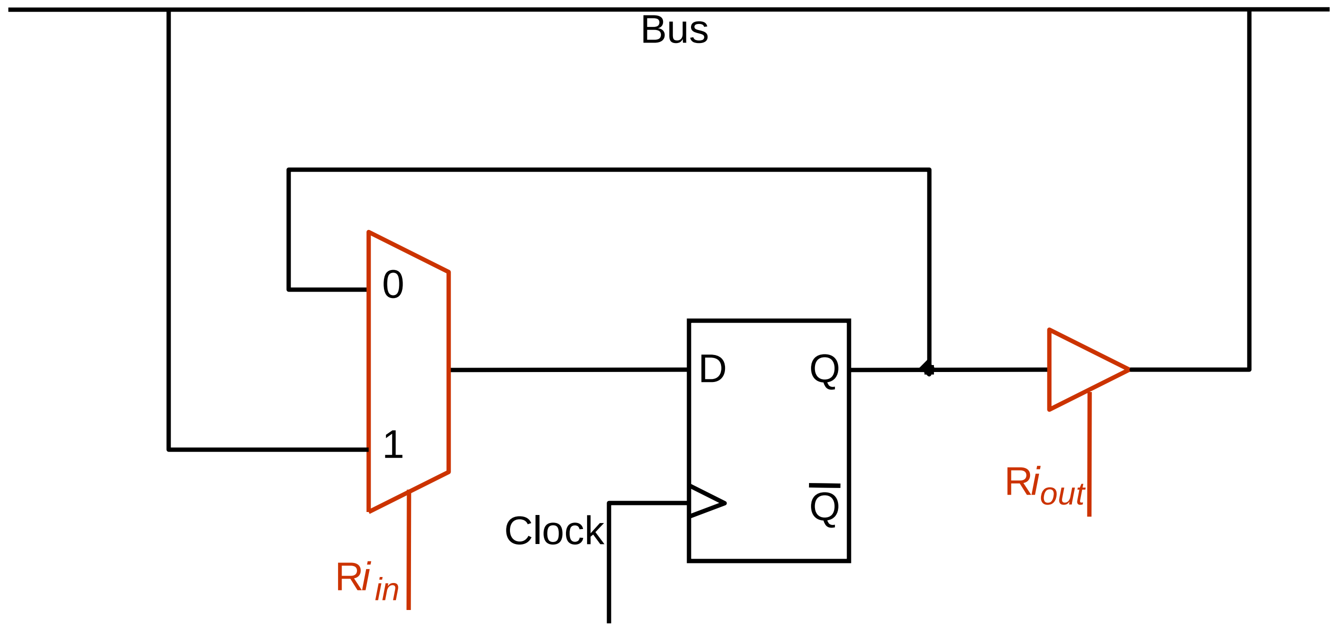
1.R1_out
2.R3_in



❖When Riin =1, data on bus is loaded into Ri

❖ when Riout =1, content of Ri is placed on bus

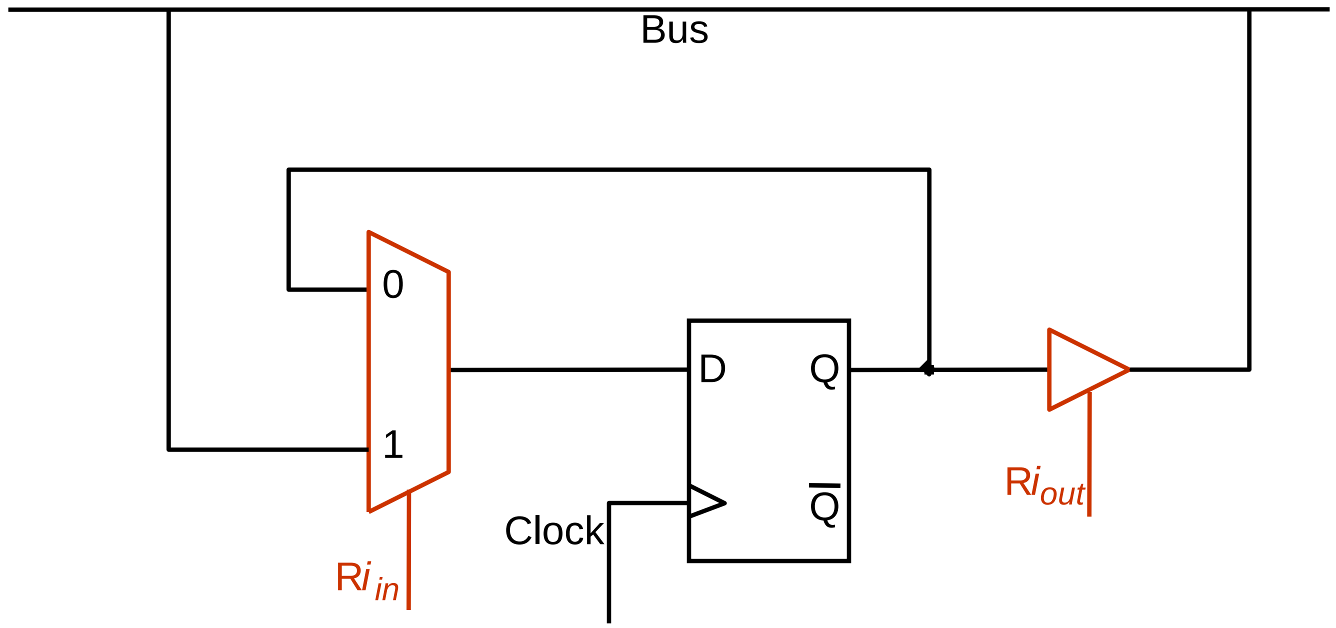
Registers and the bus (contd..)



All operations and data transfers are controlled by the processor clock

- Each bit in a register may be implemented by an edge-triggered D flip flop.
- Two input multiplexer is used to select the data applied to the input of an edge triggered flip-flop.
- Q output of the flip-flop is connected to the bus via a tri-state gate.

Registers and the bus (contd..)



Ri_{in} = 1:

Multiplexer selects the data on the bus.

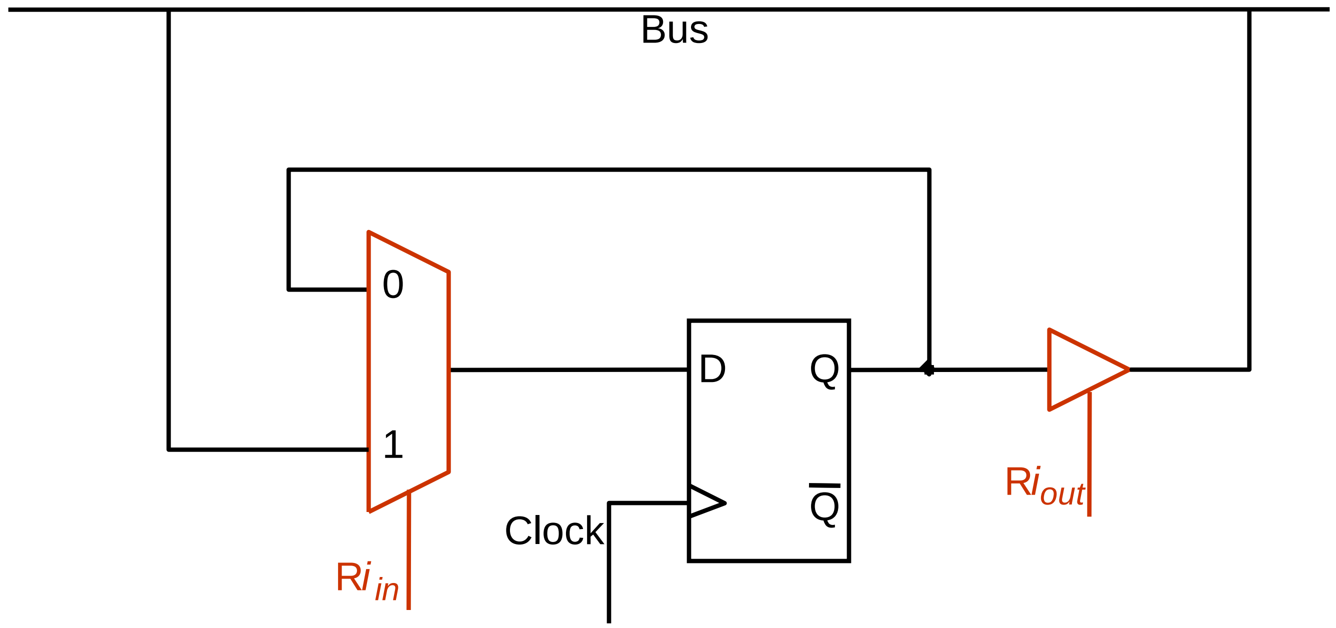
Data is loaded into the flip-flop at the rising edge of the clock.

Ri_{in} = 0:

Multiplexer feeds back the value currently stored in the flip-flop.

Q output represents the value currently stored in the flip-flop.

Registers and the bus (contd..)



Ri_{out} = 1:
Tri-state gate loads the value of the flip-flop onto the bus.
Data is loaded onto the bus at the rising edge of the clock.

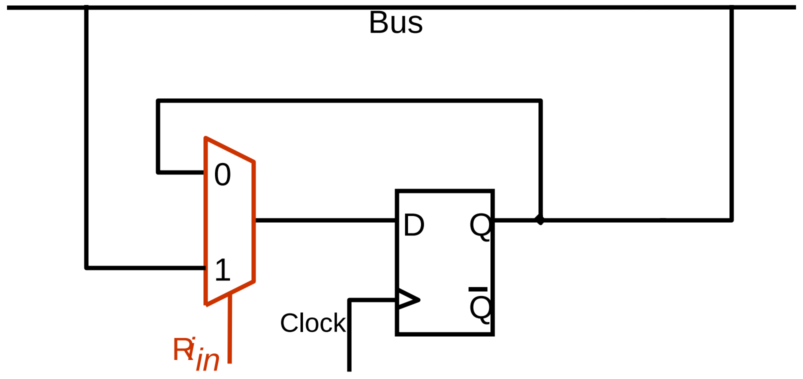
Ri_{out} = 0:
Gate's output is in high-impedance (electrically disconnected) state.
Corresponds to open-circuit state.

Registers and the bus (contd..)

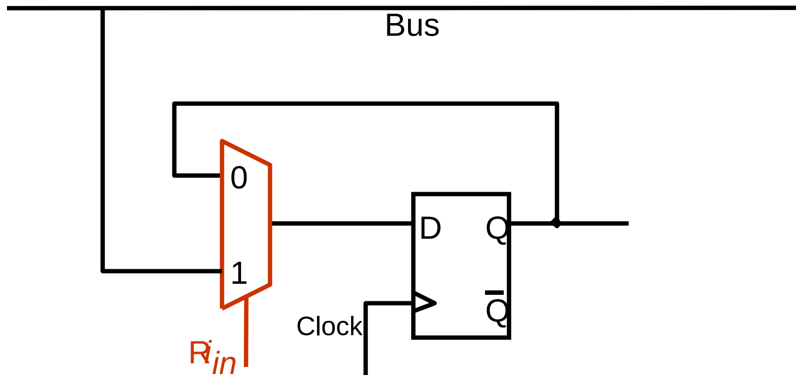
Operation of a tri-state gate

- A tri-state gate can enter one of three output states.
  - its output can be in a logic low state (L).
  - its output can be in a logic high state (H).
  - its output can be effectively an open-circuit (high impedance)
- When a tri-state gate is connected to a bus in high-impedance state, its outputs are effectively disconnected from the bus.

Ri_{out} = 1, output is:
Logic low, if Q = 0
Logic high, if Q = 1



Ri_{out} = 0:
High impedance
Open circuit condition

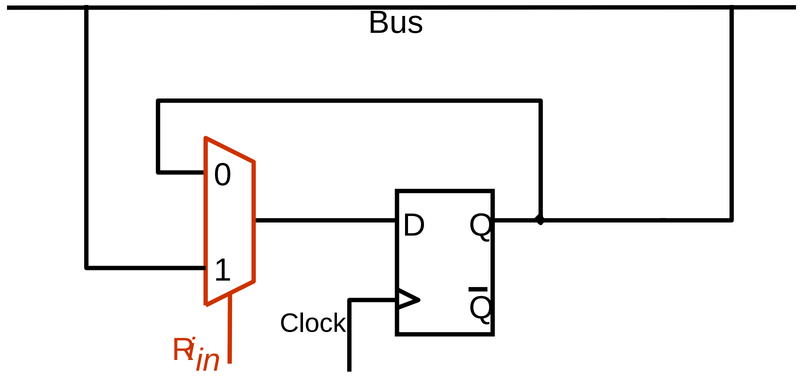


Registers and the bus (contd..)

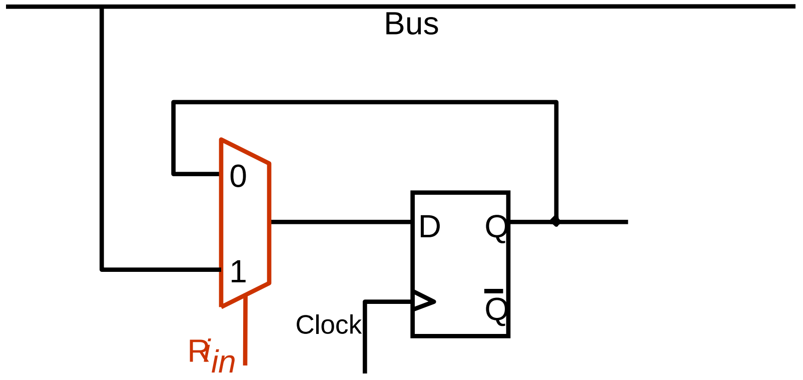
Operation of a tri-state gate

- A tri-state gate can enter one of three output states.
  - its output can be in a logic low state (L).
  - its output can be in a logic high state (H).
  - its output can be effectively an open-circuit (high impedance)
- When a tri-state gate is connected to a bus in high-impedance state, its outputs are effectively disconnected from the bus.

Ri_{out} = 1, output is:
Logic low, if Q = 0
Logic high, if Q = 1

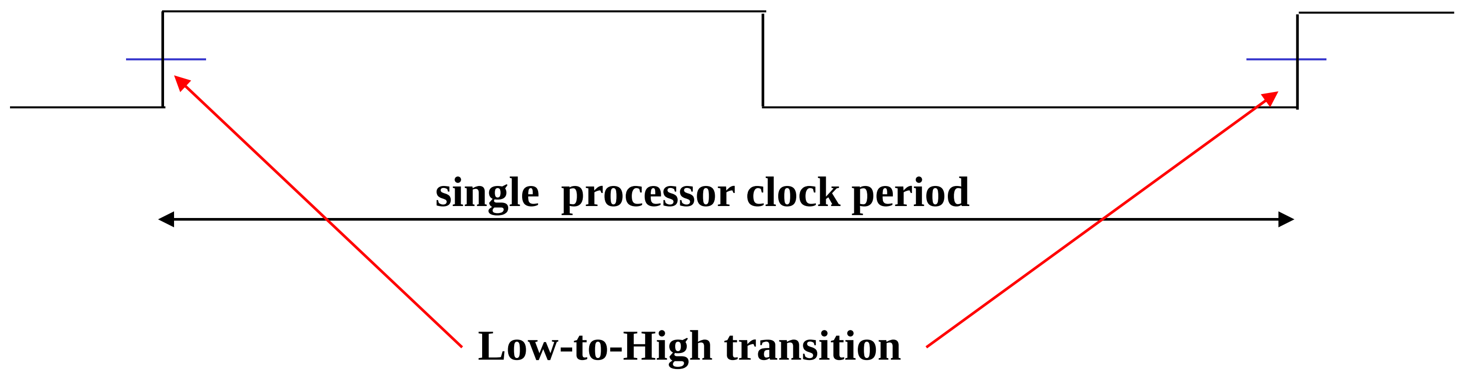


Ri_{out} = 0:
High impedance
Open circuit condition



Registers and the bus (contd..)

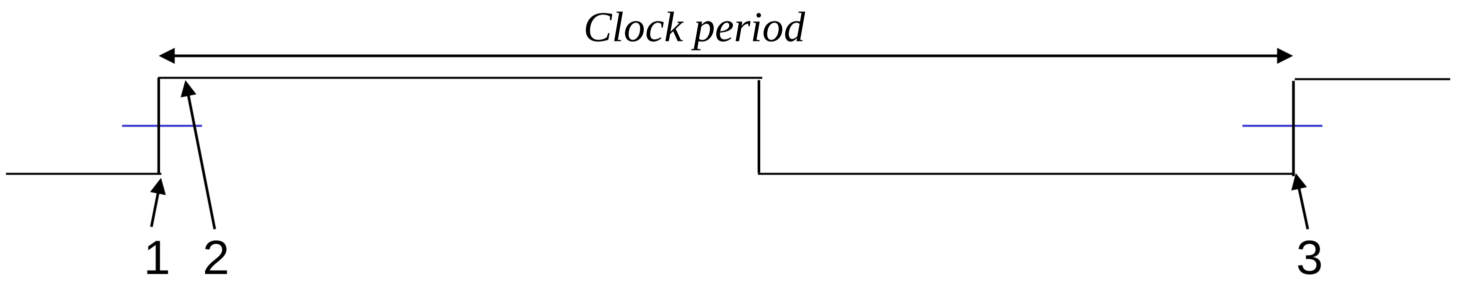
Operation of an edge-triggered flip-flop



- Data is loaded from the register to the bus (or to the register from the bus) at the rising edge of the clock.
- Data is loaded at the L-H transition of the clock.

Simple register transfer example

Transfer the contents of register R3 to register R4



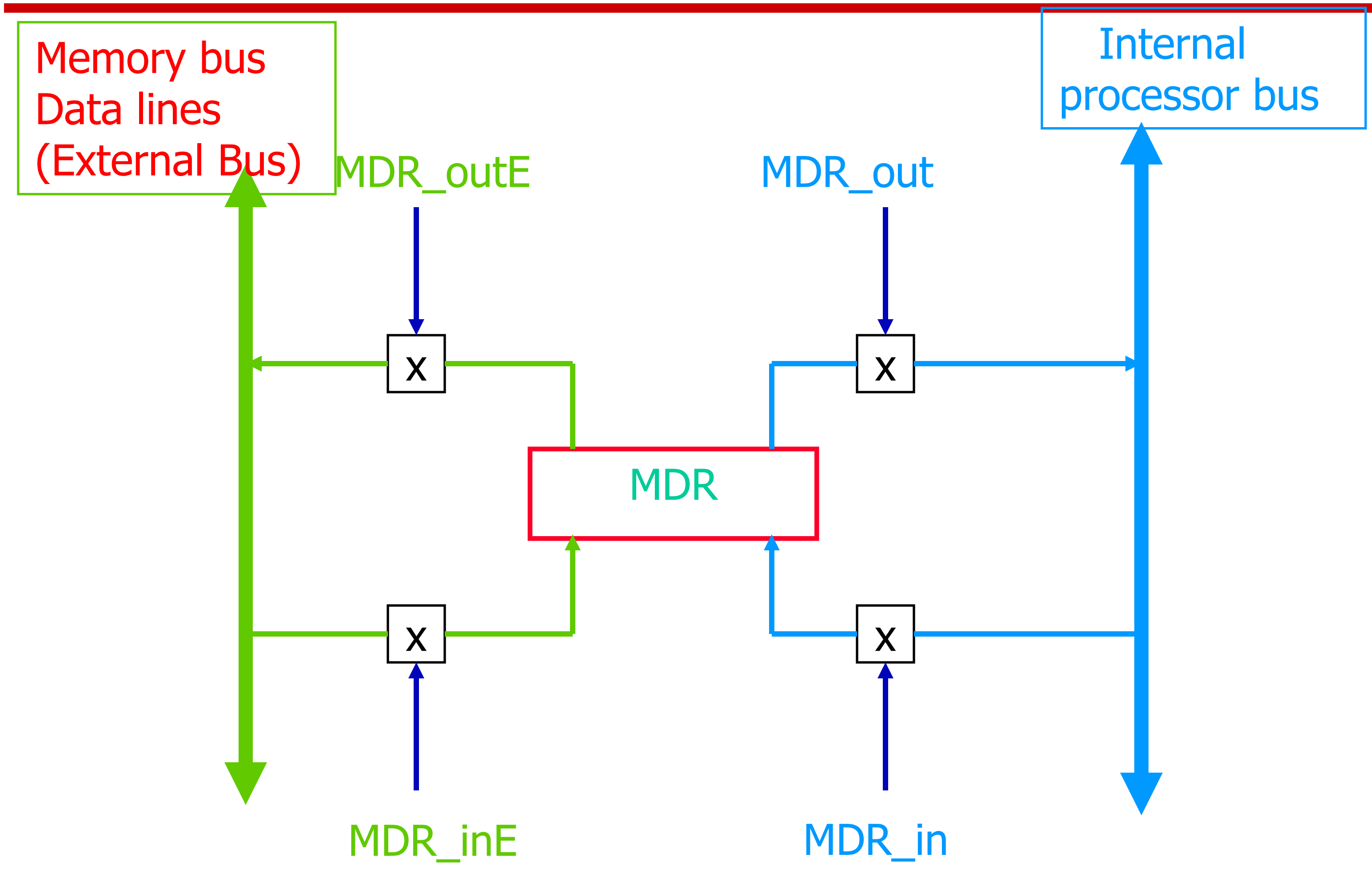
1. Control signals $R3_{out}$ and $R4_{in}$ become 1. They stay valid until the end of the clock cycle.
2. After a small delay, the contents of R3 are placed onto the bus. The contents of R3 stay onto the bus until the end of the clock cycle.
3. At the end of the clock cycle, the data onto the bus is loaded into R4. $R3_{out}$ and $R4_{in}$ become 0.

The Processing Unit

1. Basic Processing Cycle
2. Types of Operations
3. Control Mechanisms

1. Register Transfer
2. Fetch from Memory
3. Store to Memory
4. Arithmetic/Logic Ops.
5. Execution of Complete Instruction
6. Branching Ops.

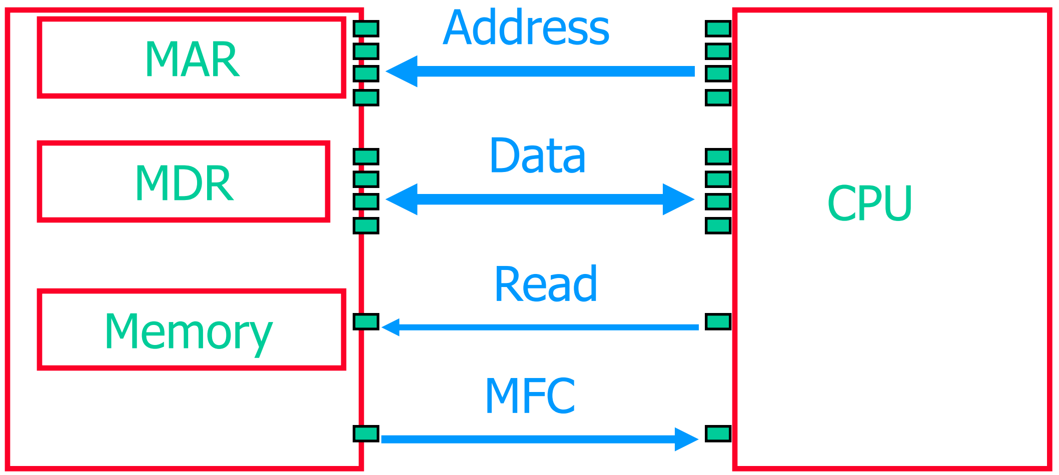
2.2. Fetch from Memory (1)



2.2. Fetch from memory (2)

e.g., Move (Ri),Rj

Control Step 1	1.	MAR ← [Ri]
	2.	Start read on memory bus
Control Step 2	3.	Wait for MFC response ← Memory Function Complete
	4.	Load MDR from memory bus
Control Step 3	5.	Rj ← [MDR]



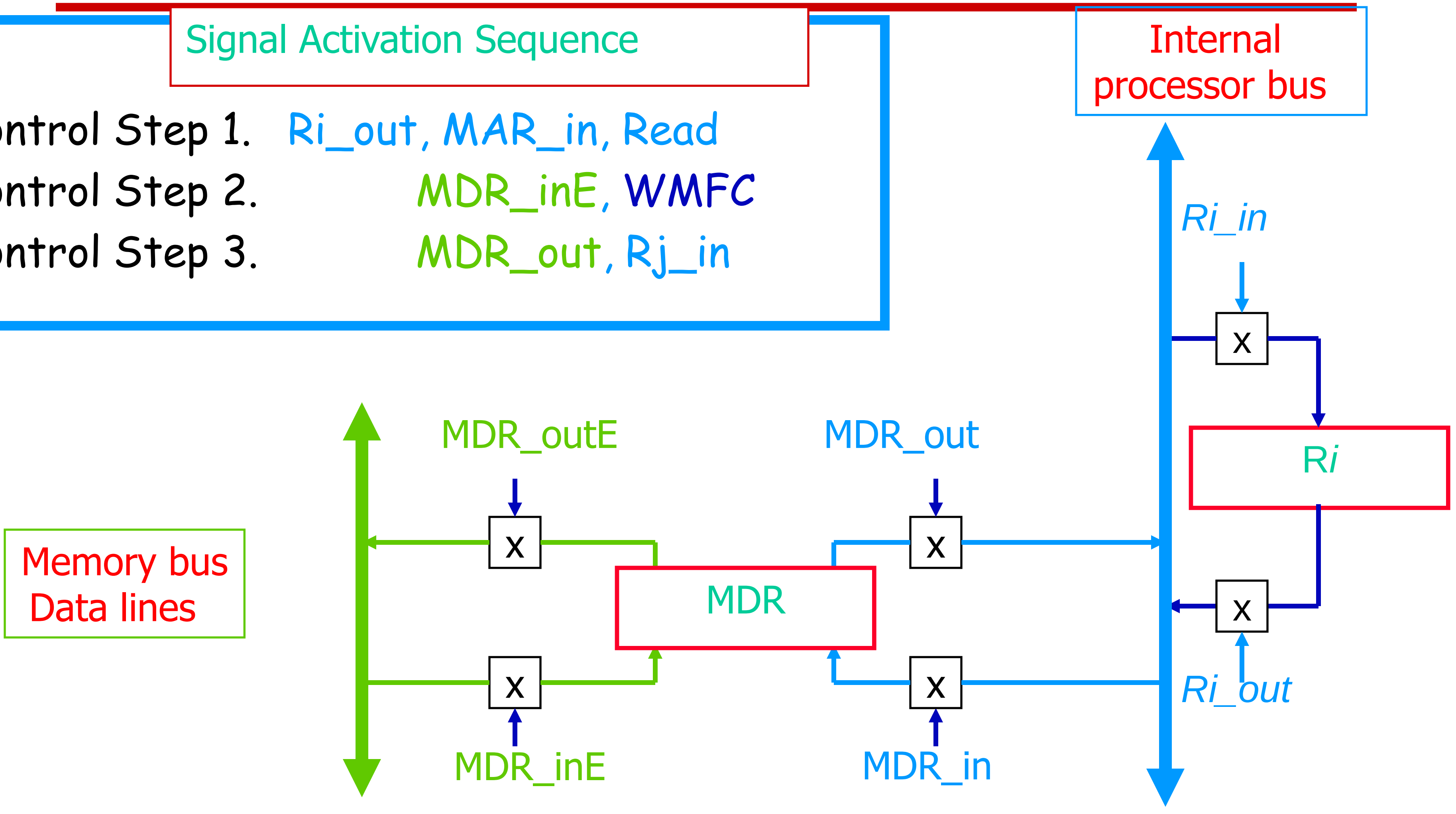
Fetch from memory (3)

Signal Activation Sequence

Control Step 1. Ri_out, MAR_in, Read

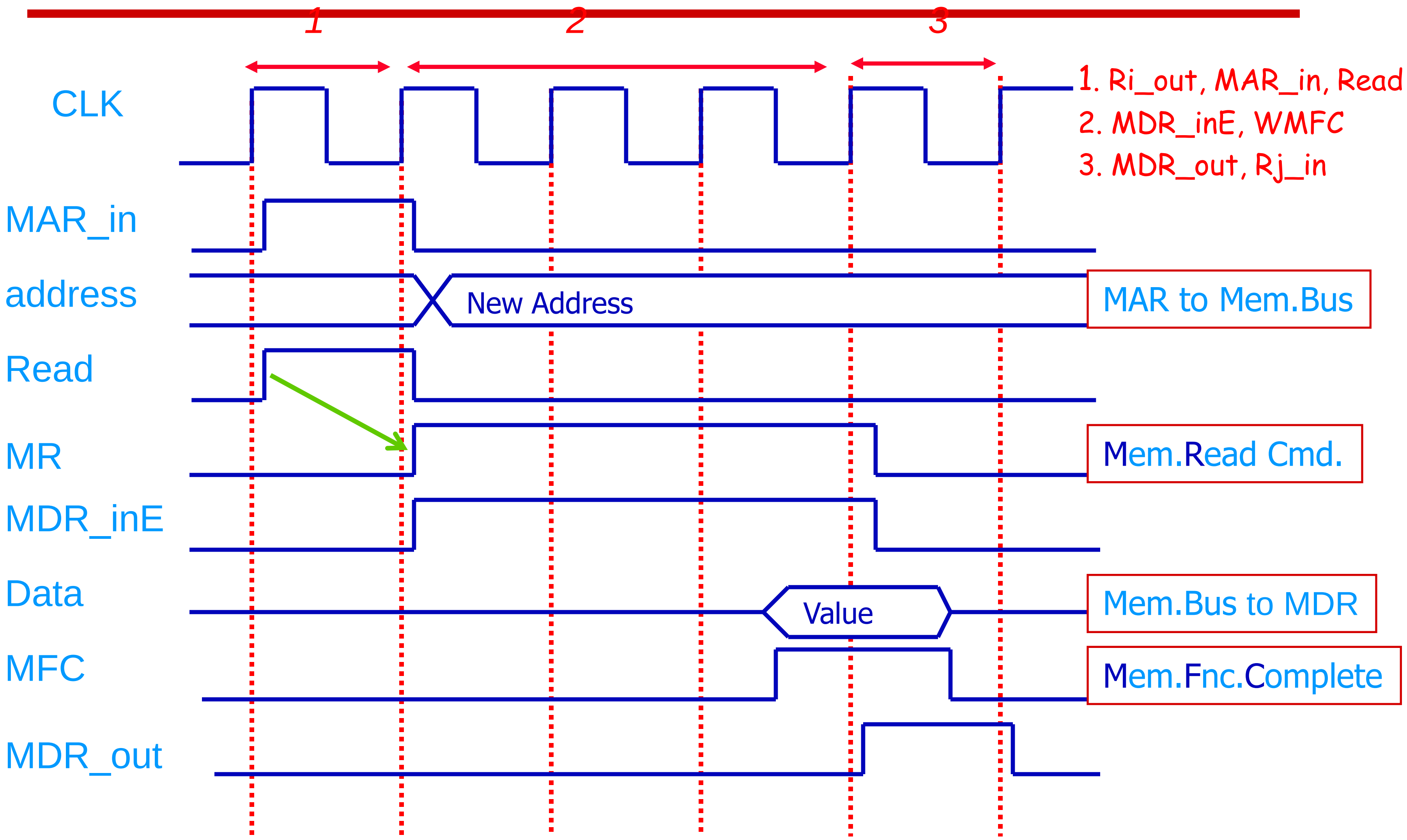
Control Step 2. MDR_inE, WMFC

Control Step 3. MDR_out, Rj_in



2.2. Fetch from Memory (4)

Timing of the Operation



1. Ri_out, MAR_in, Read
2. MDR_inE, WMFC
3. MDR_out, Rj_in

The Processing Unit

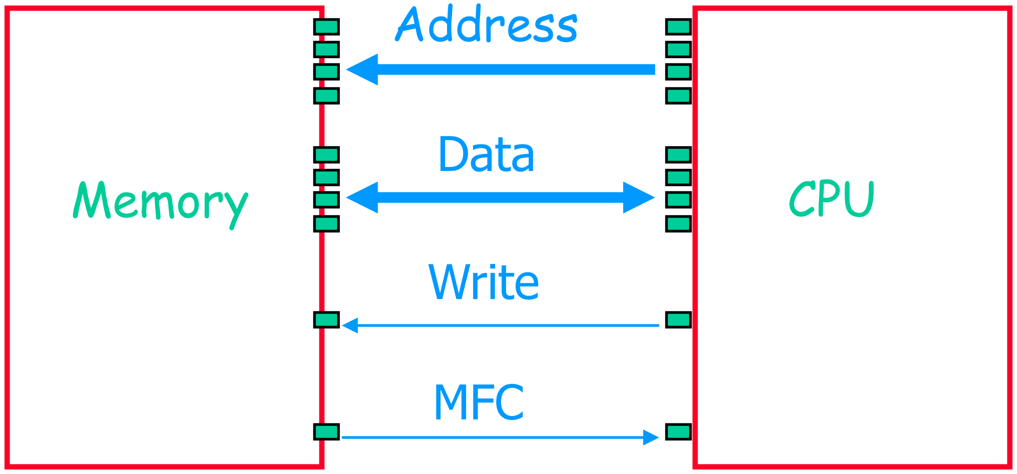
1. Basic Processing Cycle
2. Types of Operations
3. Control Mechanisms

1. Register Transfer
2. Fetch from Memory
3. Store to Memory
4. Arithmetic/Logic Ops.
5. Execution of Complete Instruction
6. Branching Ops.

2.3. Store into Memory

e.g., Move Rj,(Ri)

1. Ri_out, MAR_in
2. Rj_out, MDR_in, Write
3. MDR_outE, WMFC



The Processing Unit

1. Basic Processing Cycle
2. Types of Operations
3. Control Mechanisms

1. Register Transfer
2. Fetch from Memory
3. Store to Memory
4. Arithmetic/Logic Ops.
5. Execution of Complete Instruction
6. Branching Ops.

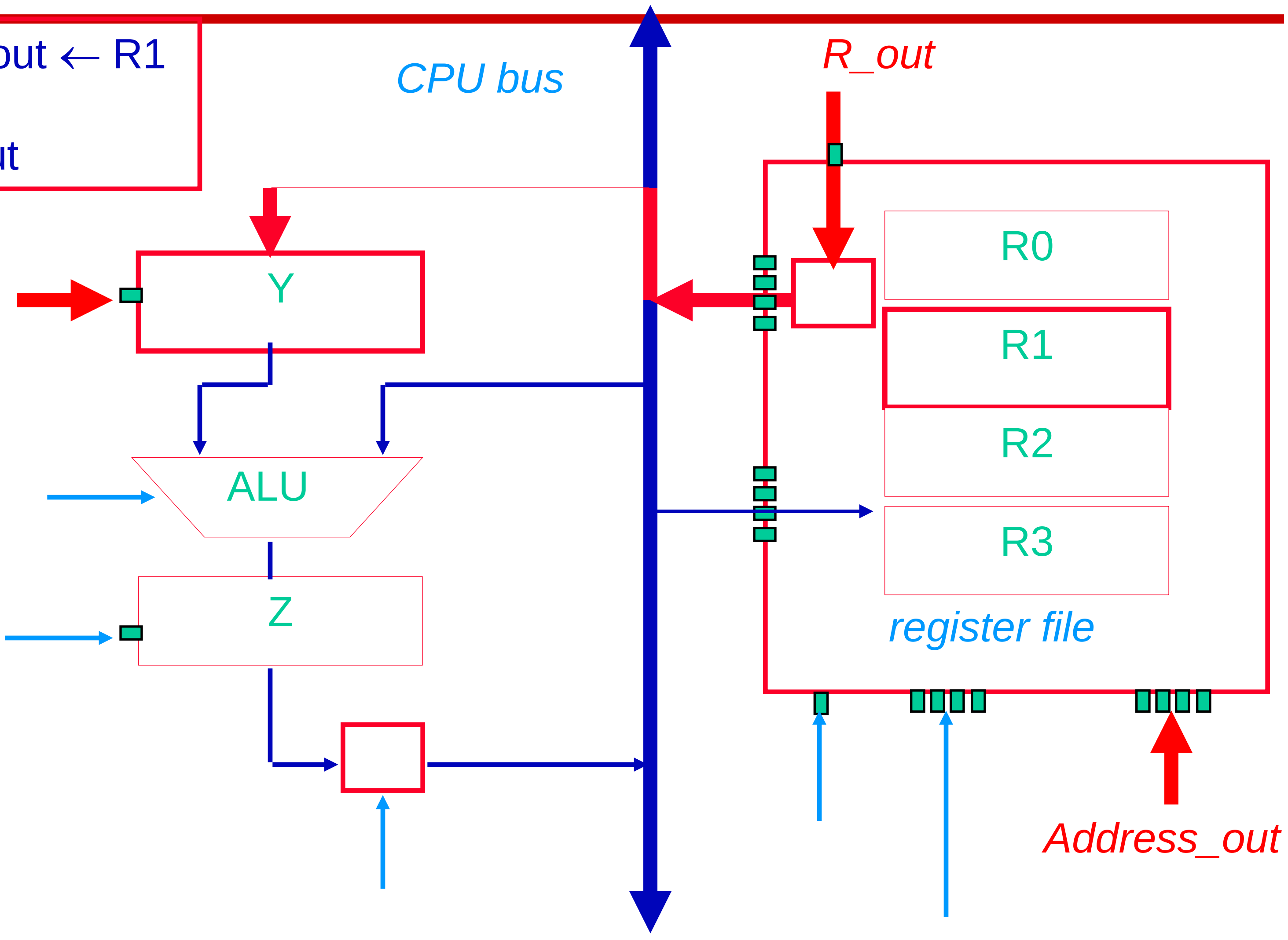
2.4. Arithmetic Operation

`ADD R3,R2,R1`

Step	Action
1.	Address out _ R1
	Y_in
	R_out
2.	Address out _R2
	R_out
	F_alu $\leftarrow$ "ADD"
	Z_in
□□	Address in _ R3
	Z_out
	R_in

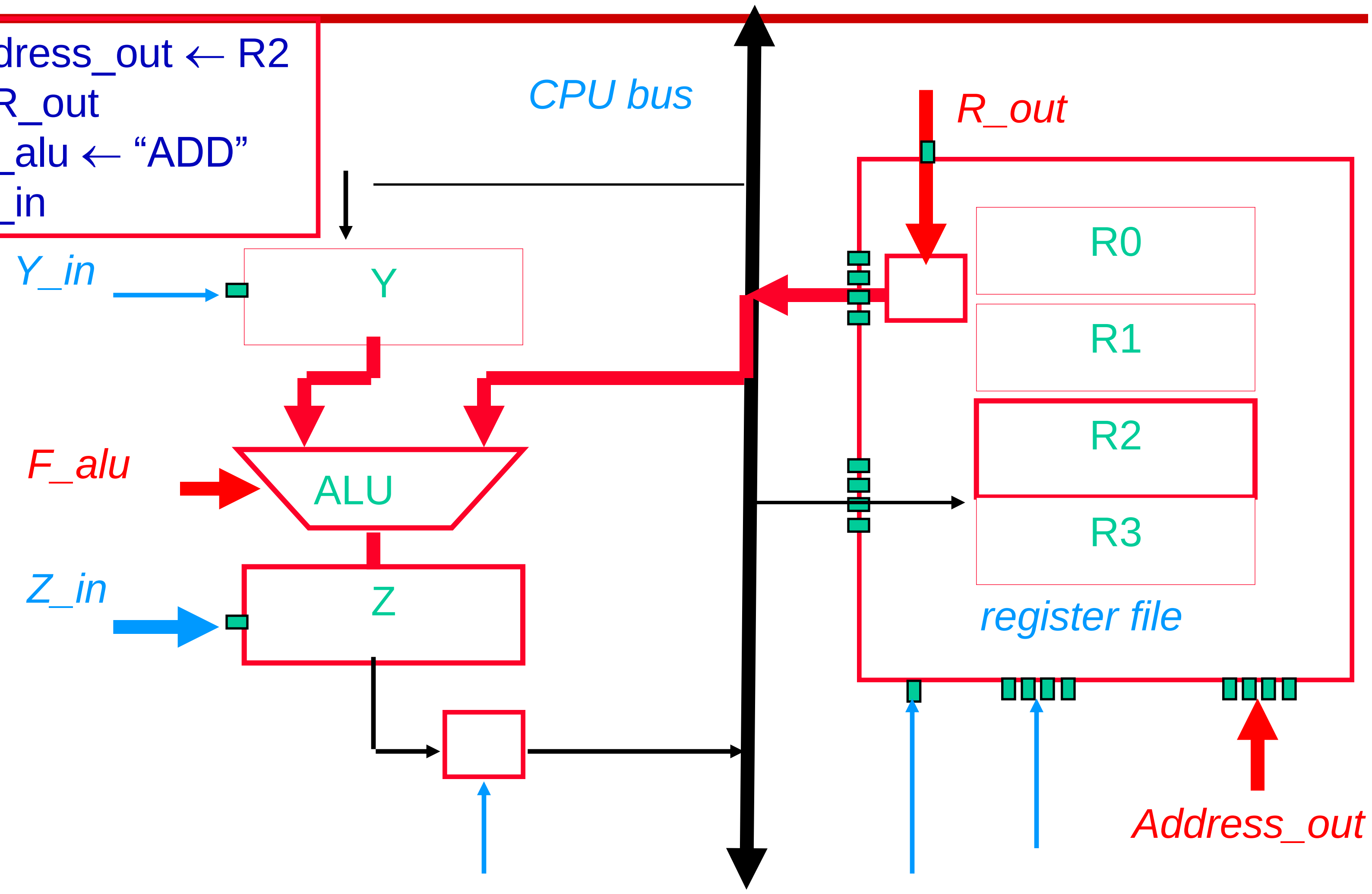
Register Transfers

1. Address_out ← R1
   Y_in
   R_out



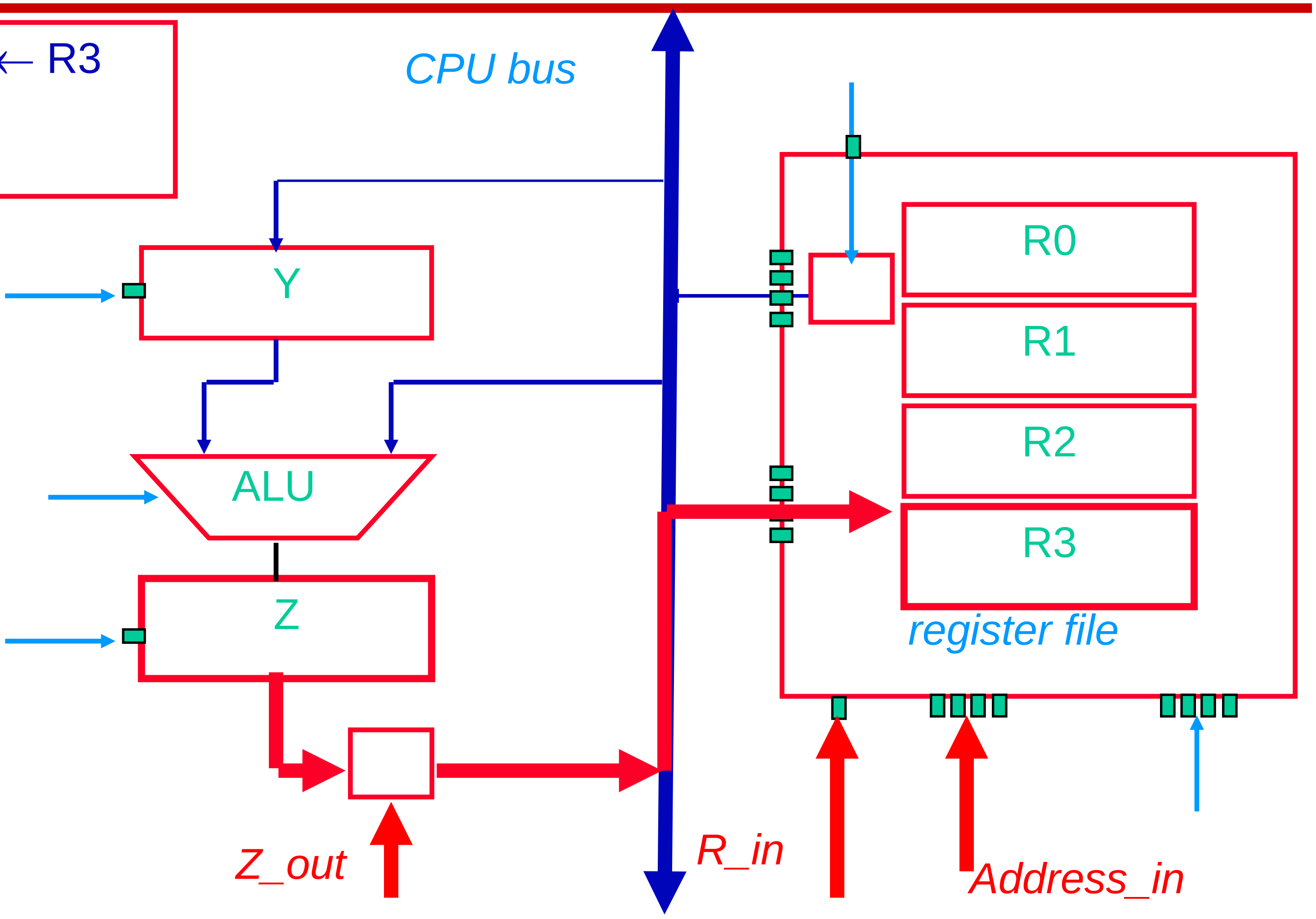
Register Transfers

2. Address_out ← R2
   R_out
   F_alu ← "ADD"
   Z_in

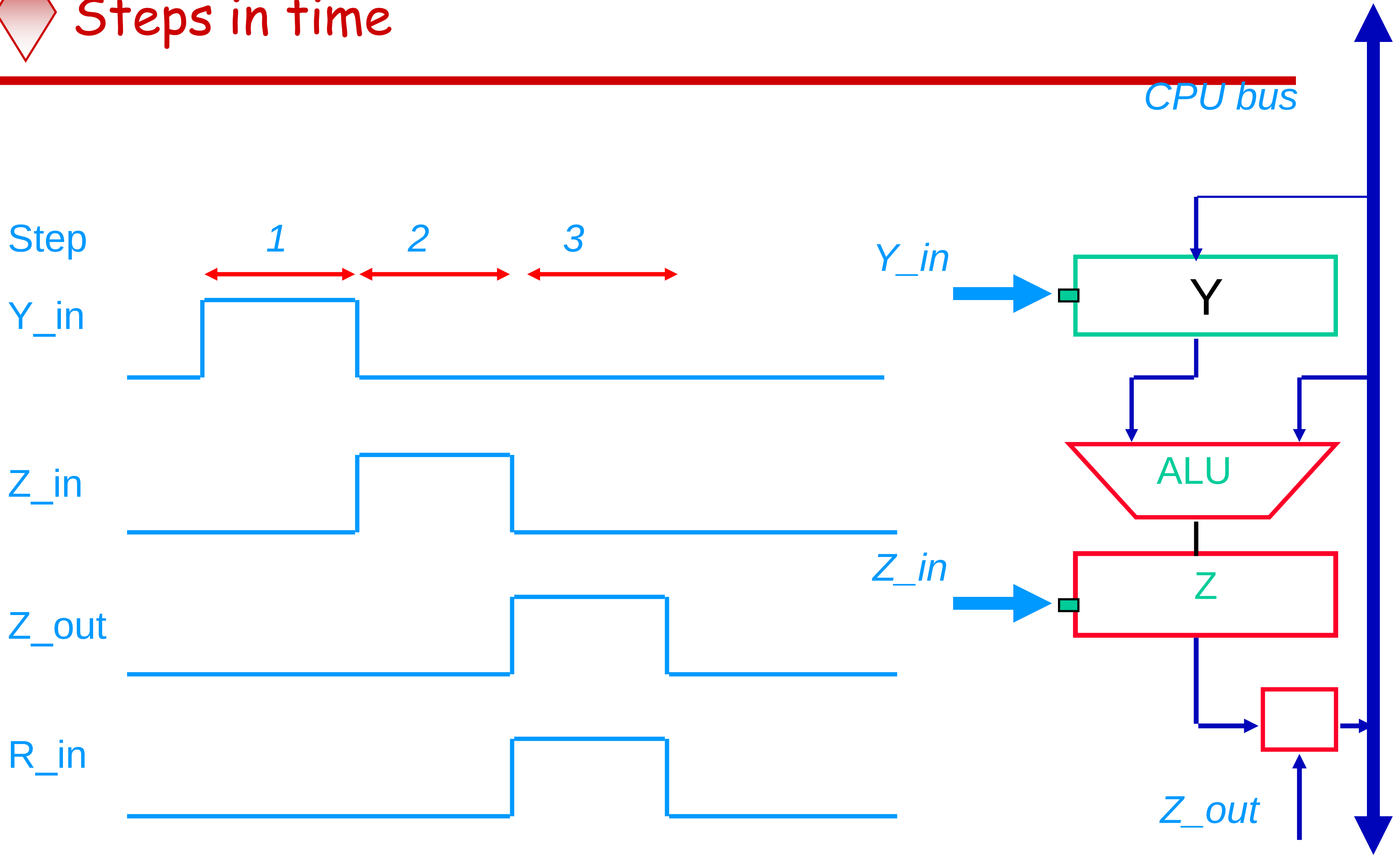


Register Transfers

3.Address_in ← R3
Z_out
R_in



Steps in time



The Processing Unit

1. Basic Processing Cycle
2. Types of Operations
   1. Register Transfer
   2. Fetch from Memory
   3. Store to Memory
   4. Arithmetic/Logic Ops.
   5. Execution of Complete Instruction
   6. Branching Ops.
3. Control Mechanisms

2.5. Execution of a Complete Instruction

1. Fetch instruction
2. Fetch the operand
3. Perform operation
4. Store result

- Example ADD (R3),R1

  [R1] ← M([R3]) + [R1]

Execution fetch (1)

Step	Action
1	PC_out, MAR_in, Read Set carry-in ALU F_alu = “ADD” Z_in
2	Z_out, PC_in Wait for MFC
3	MDR_out, IR_in

[R1] ← M([R3]) + [R1]

Step 1-3:
Instruction fetch and PC update

$[PC] \leftarrow [PC]+1$

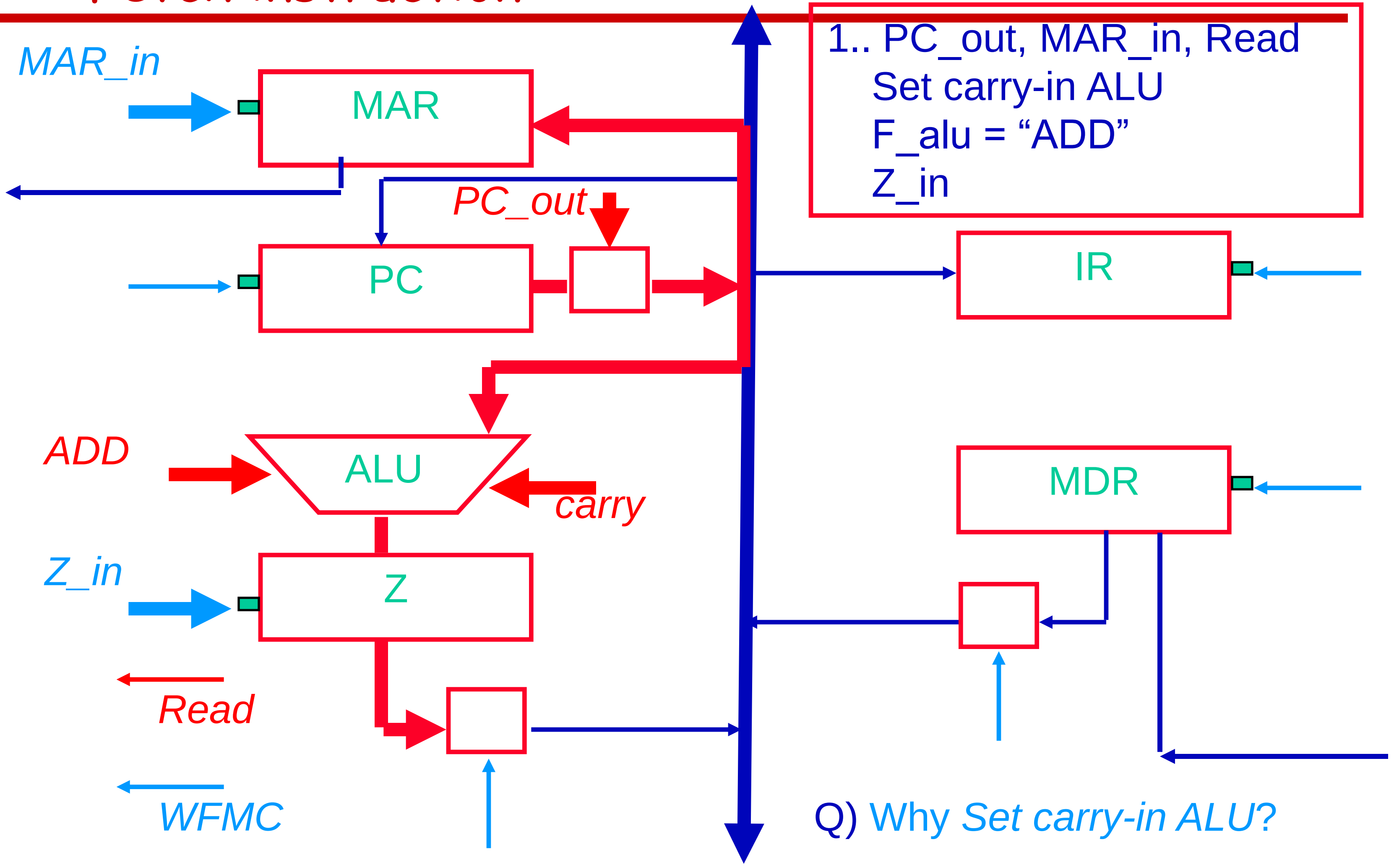
$[IR] \leftarrow M([PC])$

Note: for architectures having PC:=PC+4
a different scheme must be used

Fetch instruction

[R1] ← M([R3]) + [R1]

1.. PC_out, MAR_in, Read
Set carry-in ALU
F_alu = “ADD”
Z_in



Q) Why *Set carry-in ALU*?

Q) Why *MAR_in*?

Execution fetch (2) [R1] ← M([R3]) + [R1]

Step	Action
1	PC_out, MAR_in, Read Set carry-in ALU F_alu = “ADD” Z_in
2	Z_out, PC_in Wait for MFC
3	MDR_out, IR_in

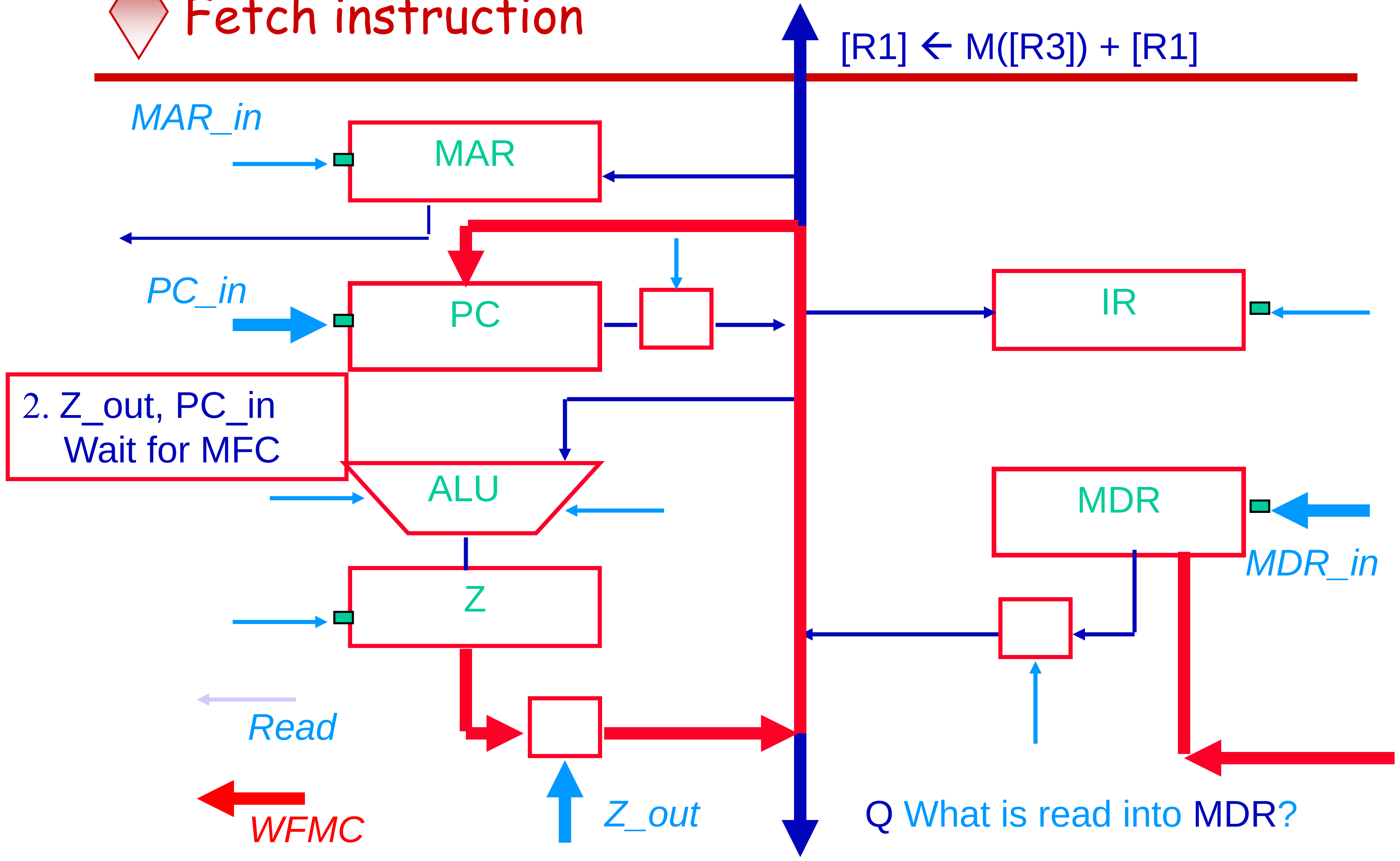
Step 1-3: instruction fetch and PC update

$[PC] \leftarrow [PC]+1$

$[IR] \leftarrow M([PC])$

Fetch instruction

[R1] ← M([R3]) + [R1]



2. Z_out, PC_in
Wait for MFC

Q What is read into MDR?

Execution fetch (3)

$[R1] \leftarrow M([R3]) + [R1]$

Step	Action
1	PC_out, MAR_in, Read Set carry-in ALU F_alu = “ADD” Z_in
2	Z_out, PC_in Wait for MFC
3	MDR_out, IR_in

Step 1-3: instruction fetch and PC update

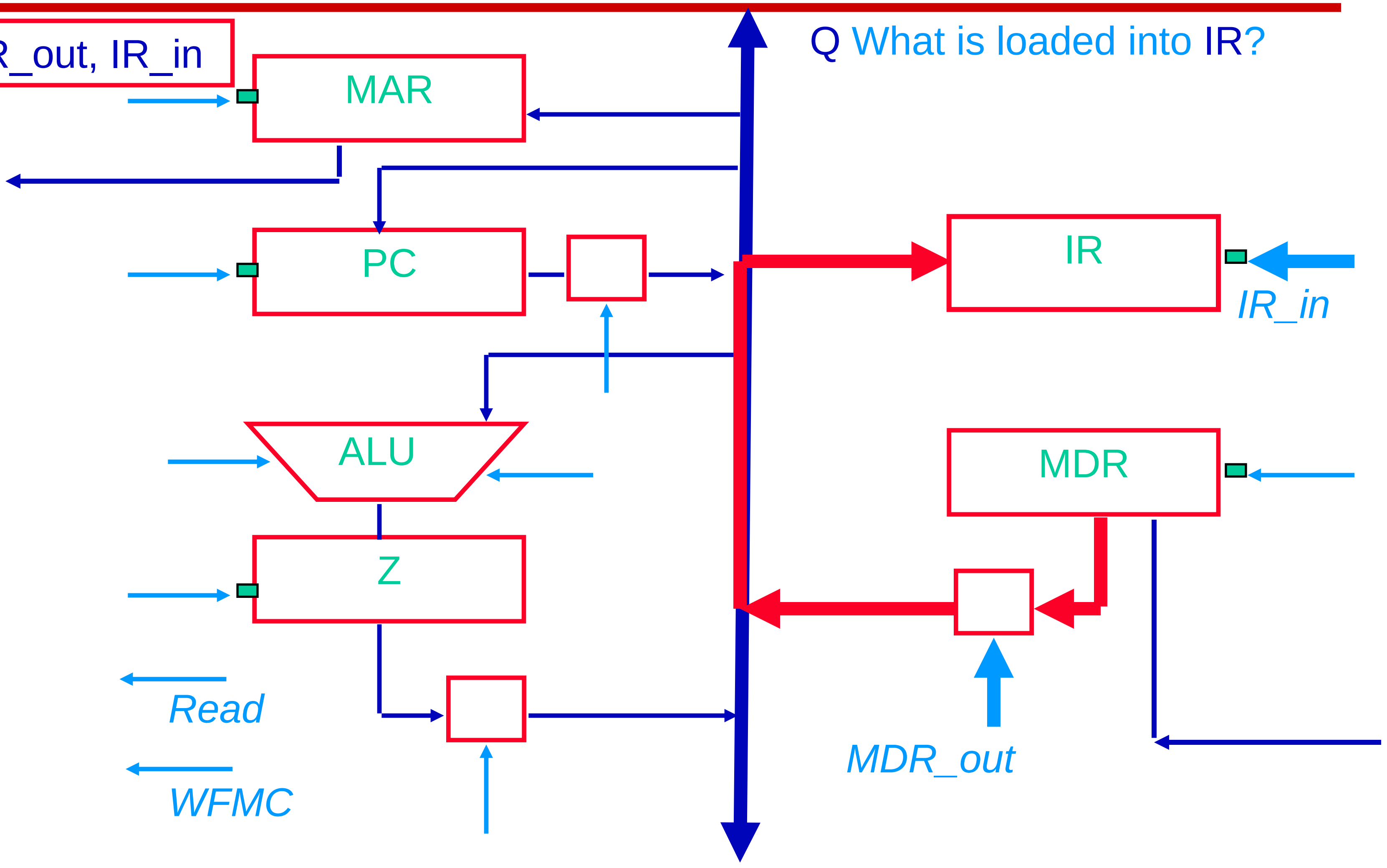
$[IR] \leftarrow M([PC])$

Fetch instruction

[R1] ← M([R3]) + [R1]

3. MDR_out, IR_in

Q What is loaded into IR?



Execute

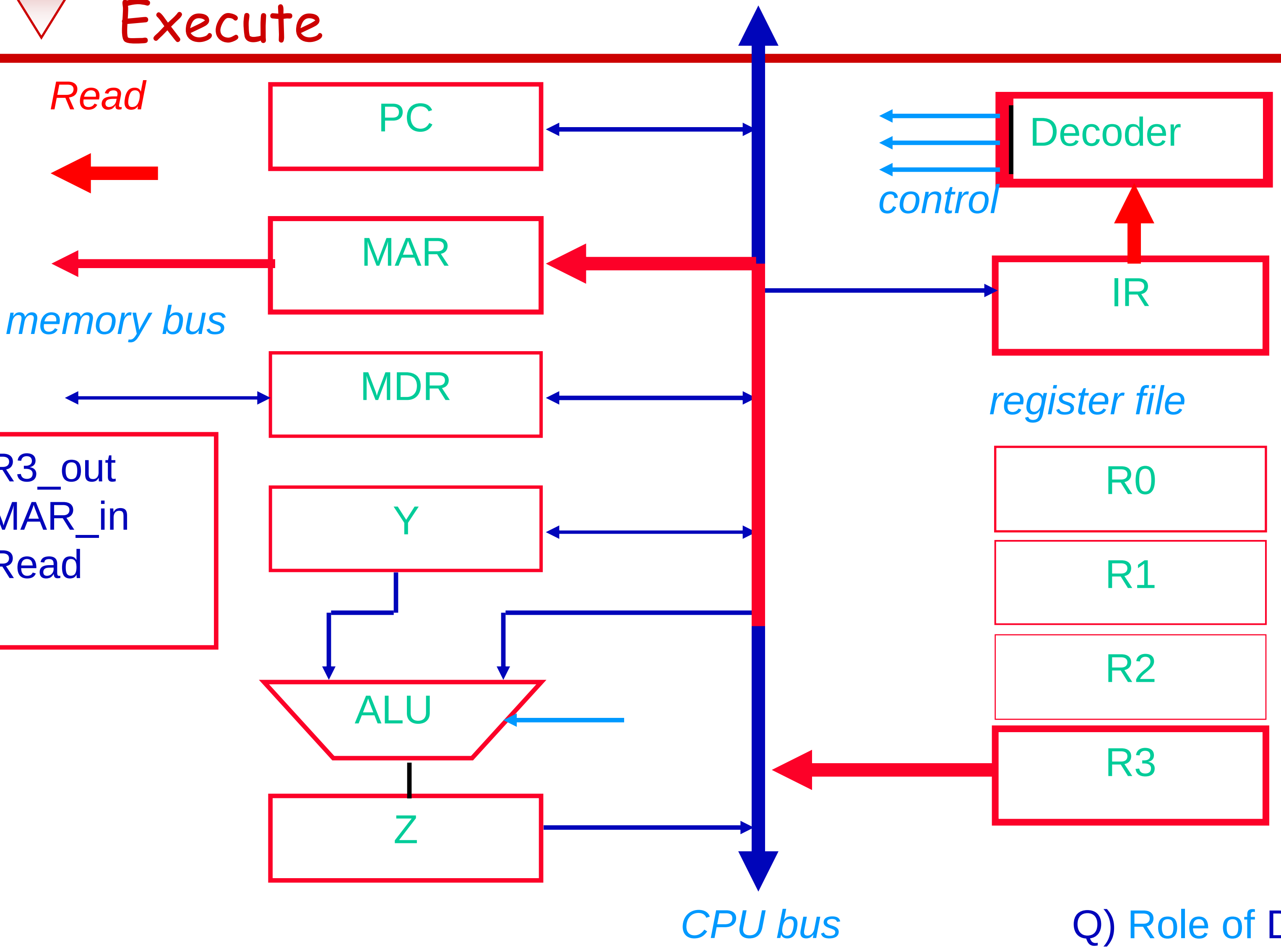
Step	Action
4	Address_out=R3, R_out MAR_in Read
5	Address_out=R1, R_out Y_in, Wait for MFC
6	MDR_out, Z_in F_alu = “ADD”
7	Address_in=R1, R_in Z_out, End

Step 4 and 5: operand fetch

Perform addition

Store Result

Execute



4. R3_out
   MAR_in
   Read

Q) Role of Decoder?

Execute

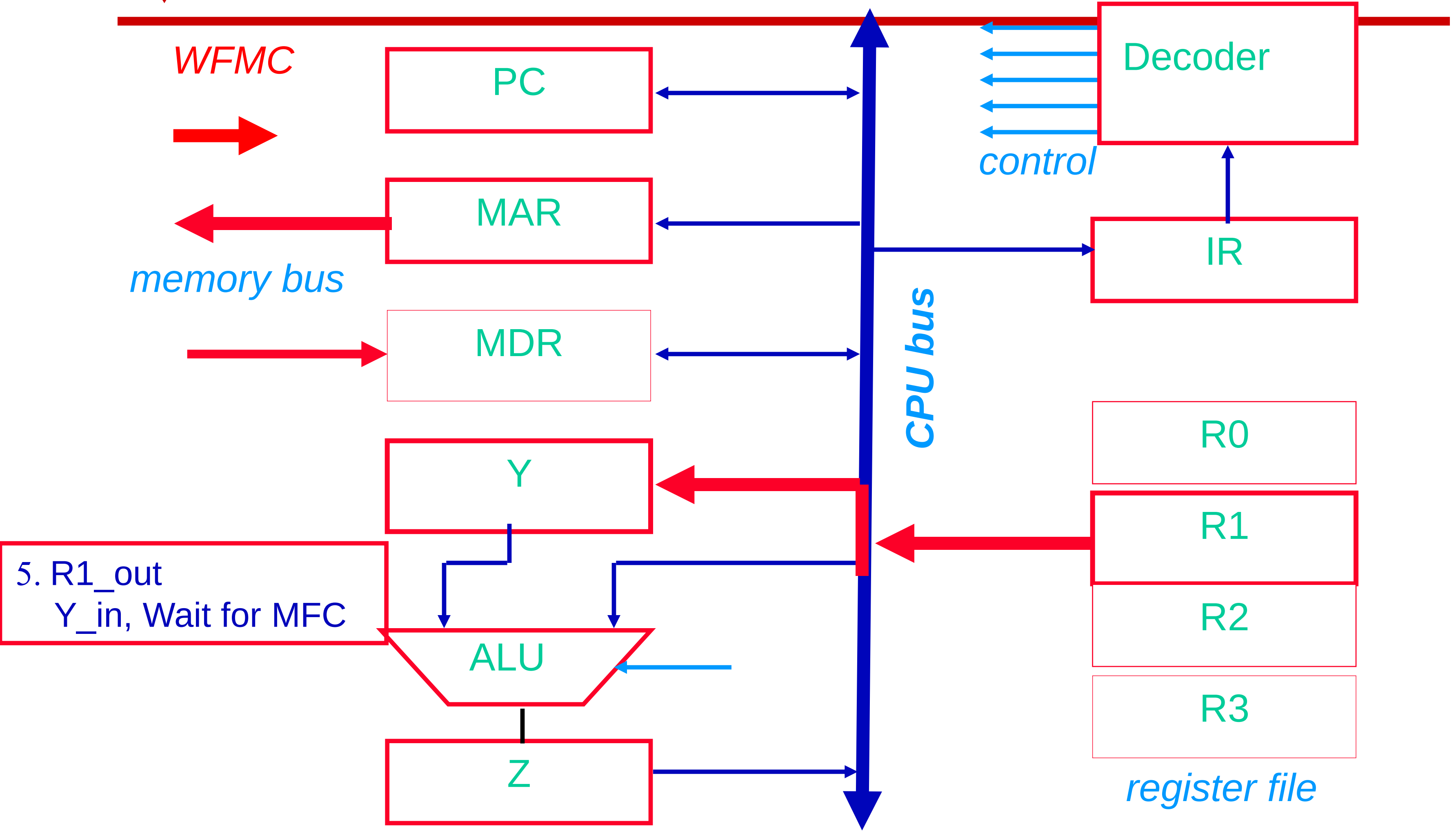
Step	Action
4	Address_out=R3, R_out MAR_in Read
5	Address_out=R1, R_out Y_in, Wait for MFC
6	MDR_out, Z_in F_alu = “ADD”
7	Address_in=R1, R_in Z_out, End

Step 4 and 5: operand fetch

Perform addition

Store Result

Execute



5. R1_out
Y_in, Wait for MFC

Q Where does MDR read from?

Execute

Step	Action
4	Address_out=R3, R_out MAR_in Read
5	Address_out=R1, R_out Y_in, Wait for MFC
→ 6	MDR_out, Z_in F_alu = "ADD"
7	Address_in=R1, R_in Z_out, End

Step 4 and 5: operand fetch

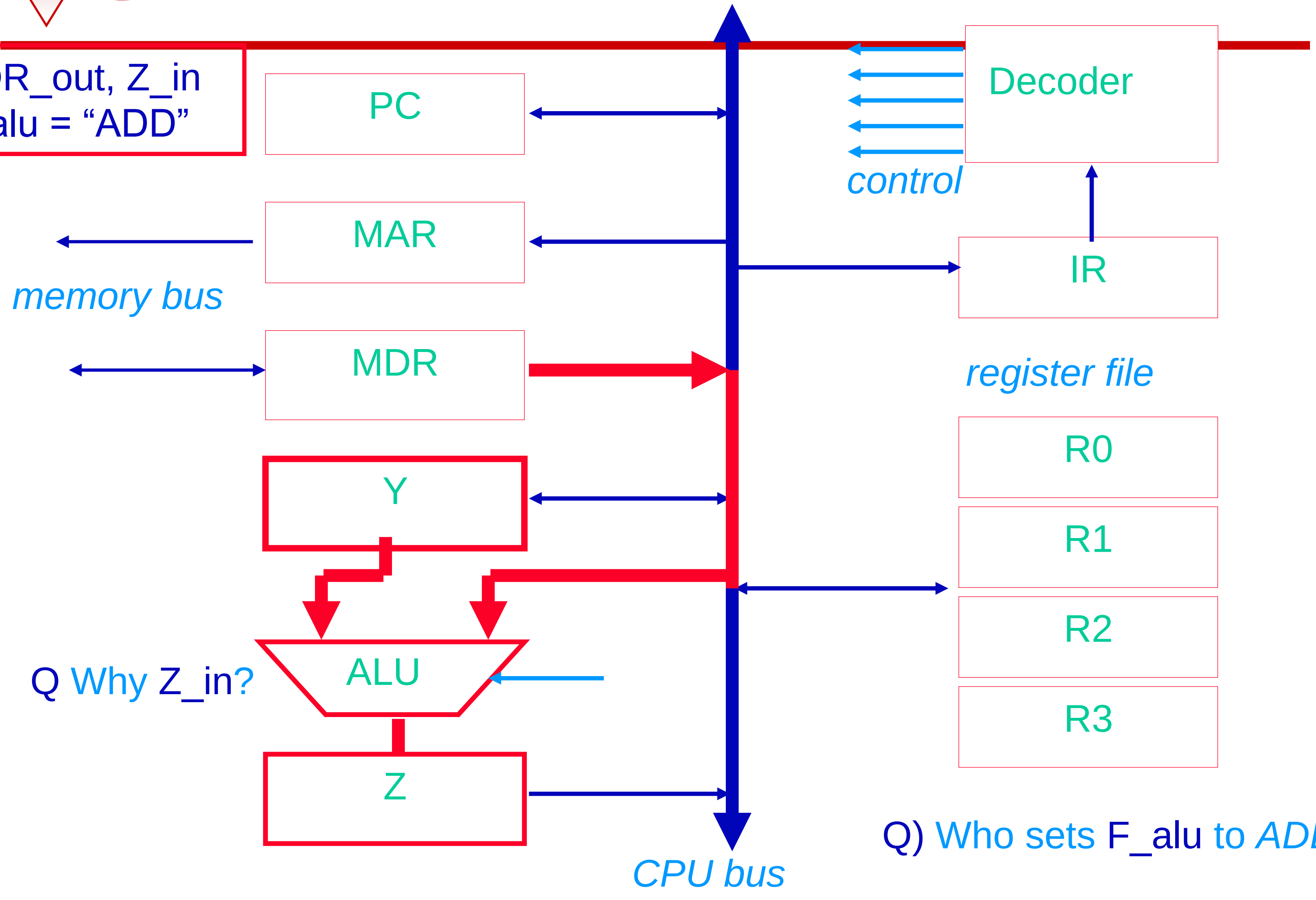
Perform addition

Store Result

Execute

[R1] ← M([R3]) + [R1]

6. MDR_out, Z_in
F_alu = "ADD"



Q Why Z_in?

Q) Who sets F_alu to *ADD*?

Execute

[R1] ← M([R3]) + [R1]

Step	Action
4	Address_out=R3, R_out MAR_in Read
5	Address_out=R1, R_out Y_in, Wait for MFC
6	MDR_out, Z_in F_alu = “ADD”
7	Address_in=R1, R_in Z_out, End

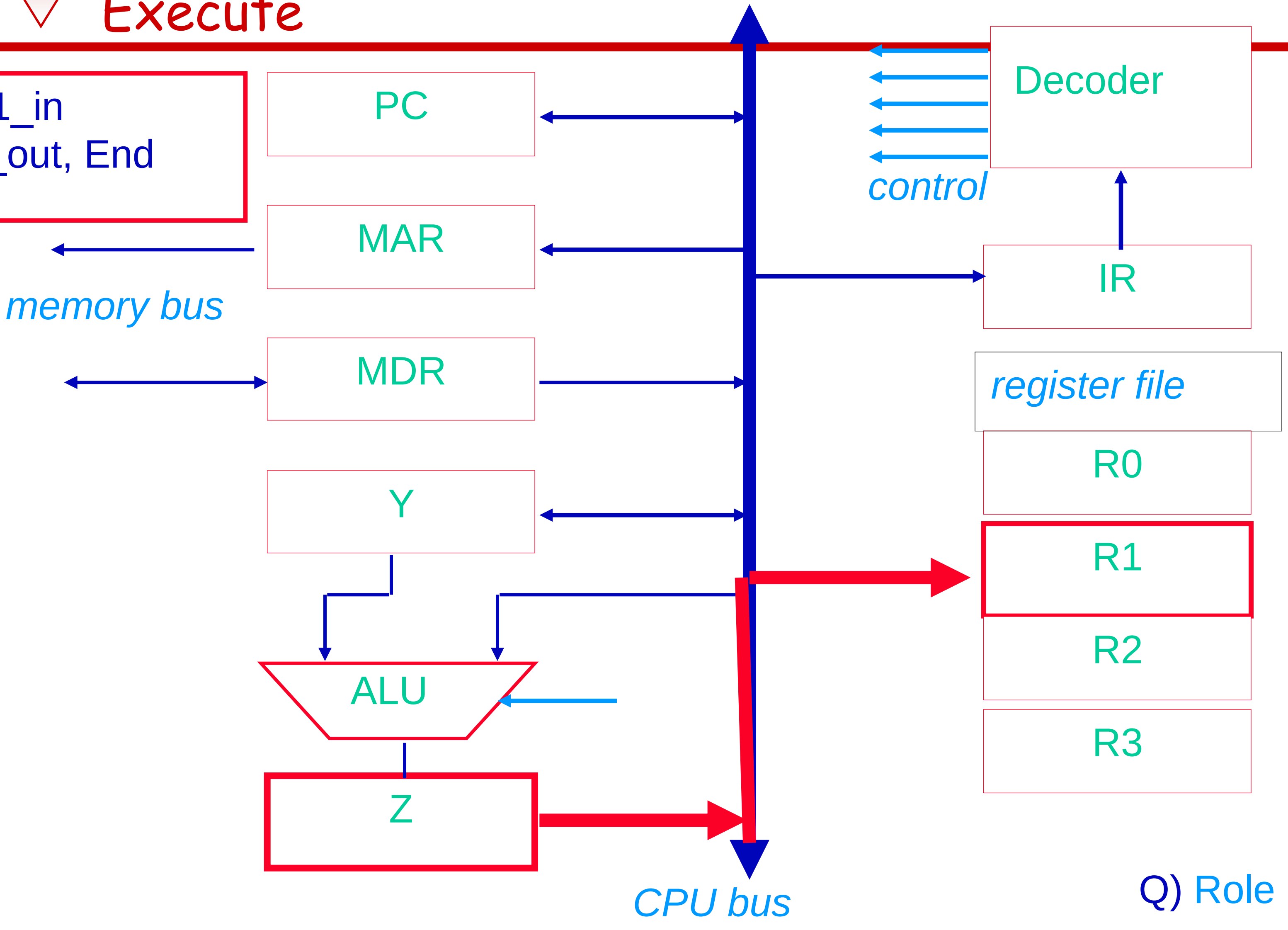
Step 4 and 5: operand fetch

Perform addition

Store Result

Execute

7. R1_in
   Z_out, End



Q) Role of End?

The Processing Unit

1. Basic Processing Cycle
2. **Types of Operations**
    1. Register Transfer
    2. Fetch from Memory
    3. Store to Memory
    4. Arithmetic/Logic Ops.
    5. Execution of Complete Instruction
    6. Branching Ops.
3. Control Mechanisms

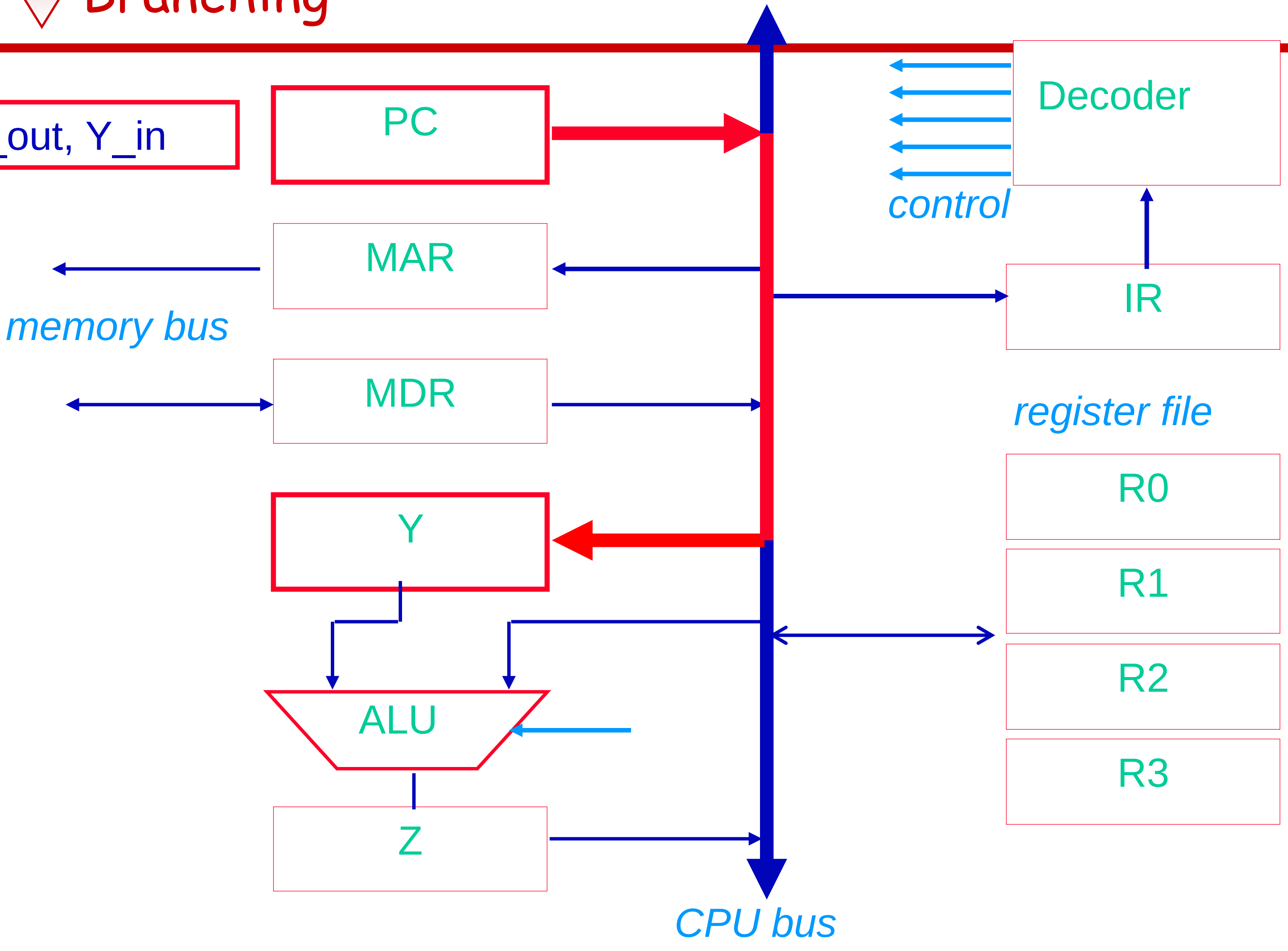
2.6. Branching

```
Jump: PC+Offset
```

Step	*Action*
1-3	<instruction fetch as in previous example>
4	PC_out, Y_in
5	Offset-field-IR_out F_alu = "ADD" Z_in
6	PC_in Z_out, End

Branching

4. PC_out, Y_in

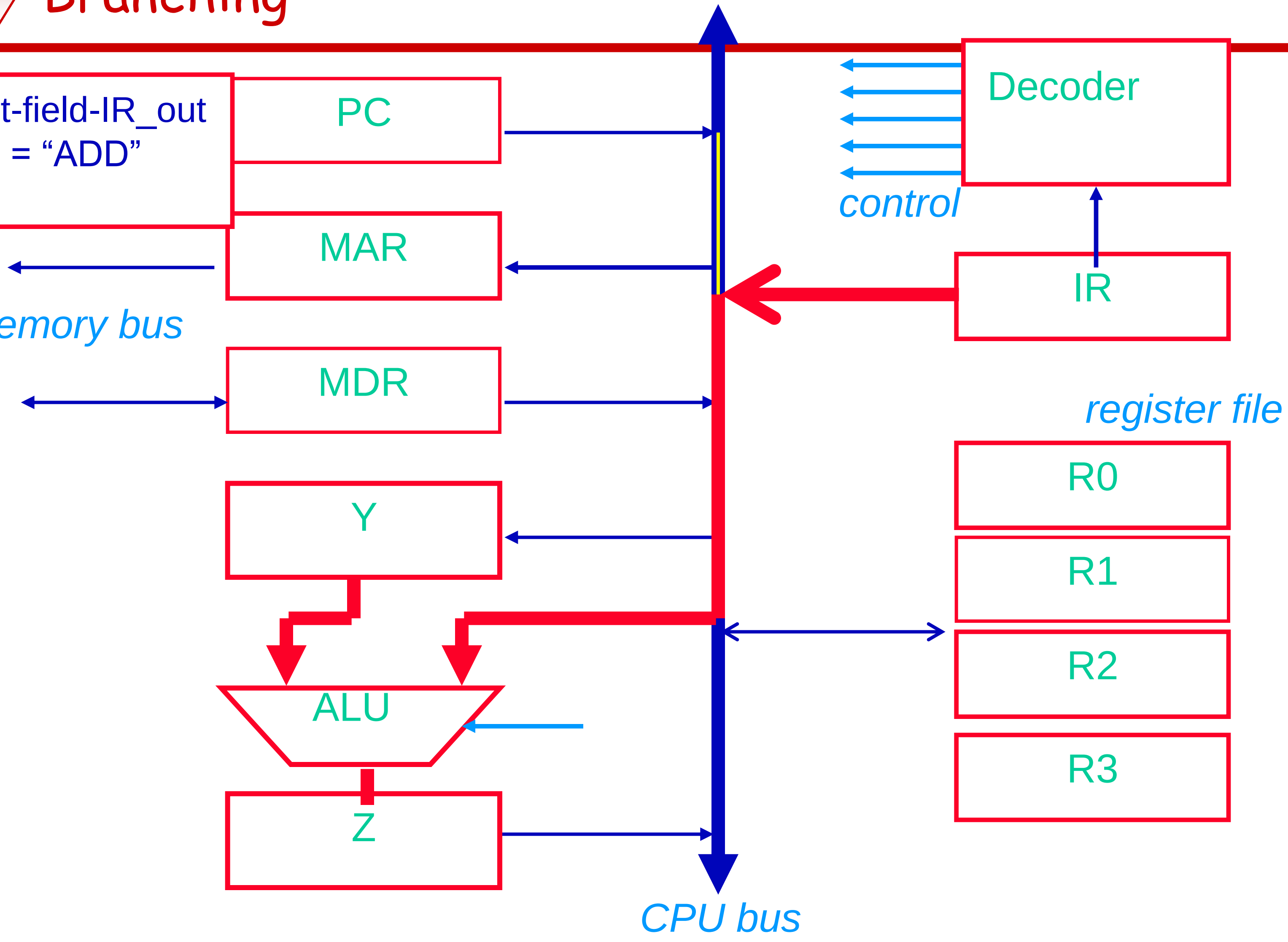


Branching

Step	*Action*
1-3	<instruction fetch as in previous example>
4	PC_out, Y_in
5	Offset-field-IR_out F_alu = “ADD” Z_in
6	PC_in Z_out, End

Branching

5. Offset-field-IR_out
   F_alu = "ADD"
   Z_in

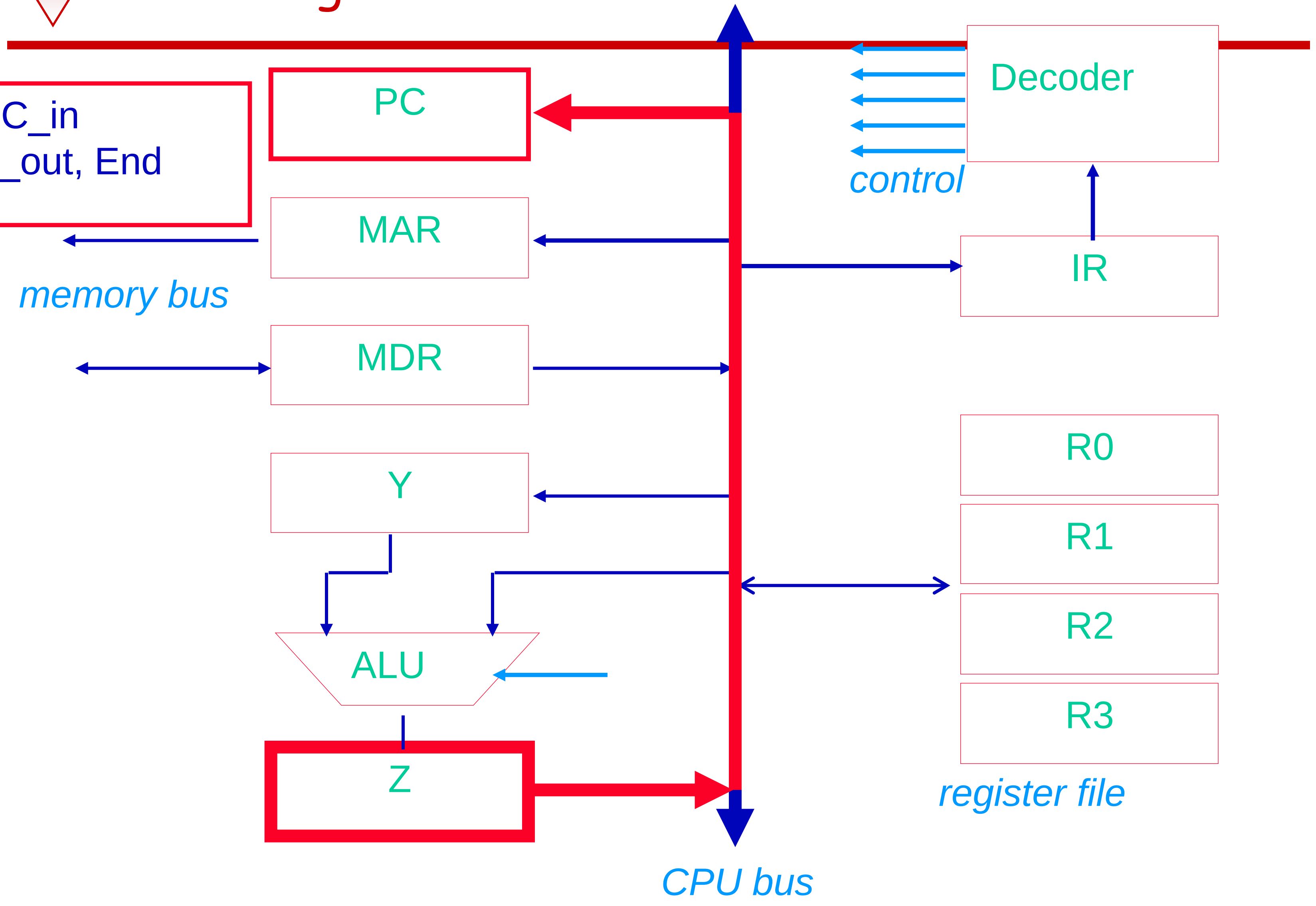


Branching

Step	*Action*
1-3	<instruction fetch as in previous example>
4	PC_out, Y_in
5	Offset-field-IR_out F_alu = "ADD" Z_in
6	PC_in Z_out, End

Branching

6. PC_in
   Z_out, End



Conditional branching

`JN : PC+Offset`

Step	Action
1-3	<instruction fetch as in previous example>
4	PC_out, Y_in If N=0 then End
5	Offset-field-IR_out F_alu = "ADD" Z_in
6	PC_in Z_out, End

If not Negative

2. Performing an Arithmetic or Logic Operation2.

Ex. $R3 \longleftarrow R1 + R2$

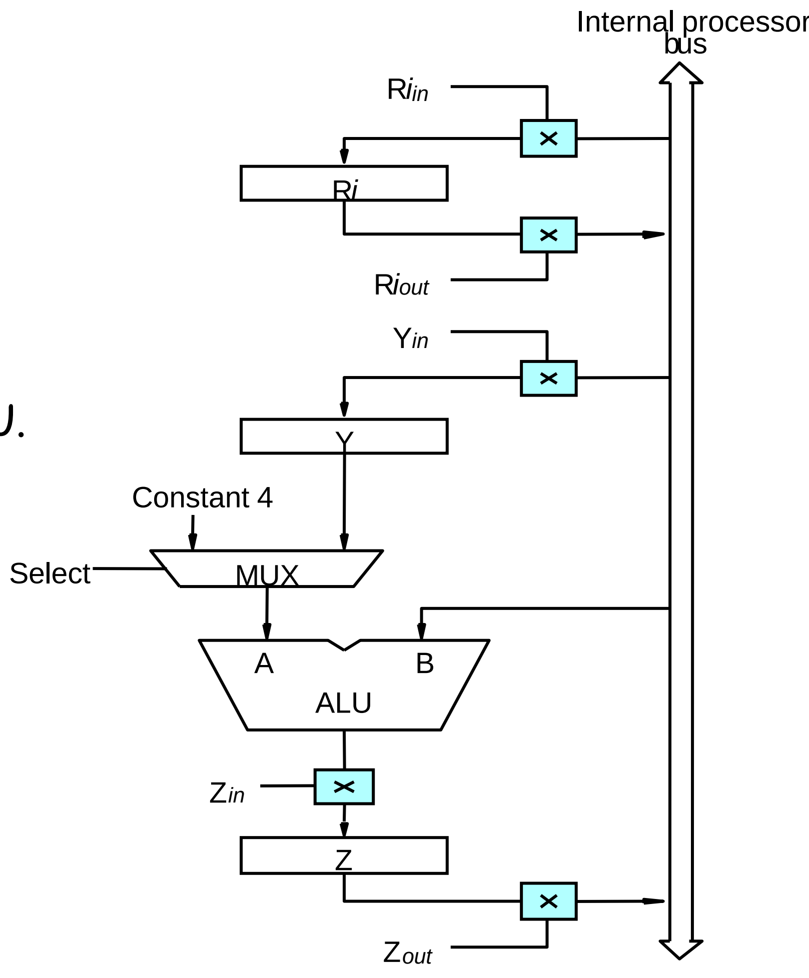
- What is the sequence of operations ?

Step 1: Output of the register R1 and input of the register Y are enabled, causing the contents of R1 to be transferred to Y.

Step 2: The multiplexer's select signal is set to select Y causing the multiplexer to gate the contents of register Y to input A of the ALU.

Step 3: The contents of Z are transferred to the destination register R3.

- R1out, Yin
- R2out, SelectY, Add, Zin
- Zout, R3in



Ex. Move (R1),R2.

The sequence of steps is:

1) R1out, MARin, Read ;desired address is loaded into MAR & Read command is issued

2) MDRinE, WMFC ;load MDR from memory bus & Wait for MFC response from memory

3) MDRout, R2in ;load R2 from MDR

where WMFC=control signal that causes processor's control circuitry to wait for arrival of MFC signal

Note: we have not considered Instruction fetch operation.

Control sequence for the instruction

EX. Move (RS),Rd

The control-sequence is written as follows

1)PCout, MARin, Read, Select4, Add, Zin
2)Zout, PCin, Yin, WMFC
3)MDRout, IRin
4)RSin, MARin, Read
5)MDRinE, WMFC
6)MDRout, Rdin, End

Execution of a complete instruction

Consider the instruction *Add (R3),R1*
Which adds he contents of a memory-
Location pointed by R3 to register R1.

Executing this instruction requires
the following actions:

1) Fetch the instruction.
2) Fetch the first operand.
3) Perform the addition.
4) Load the result into R1.

Instruction Execution

Instruction execution proceeds as follows: R1← R1 + [R3]

Step1:The instruction-fetch operation is initiated by loading contents of PC into MAR & sending a Read request to memory. The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC"s content), and the result is stored in Z

Step2: Updated value in Z is moved to PC.

Step3: Fetched instruction is moved into MDR and then to IR.

Step4: Contents of R3 are loaded into MAR & a memory read signal is issued.

Step5: Contents of R1 are transferred to Y to Prepare for addition.

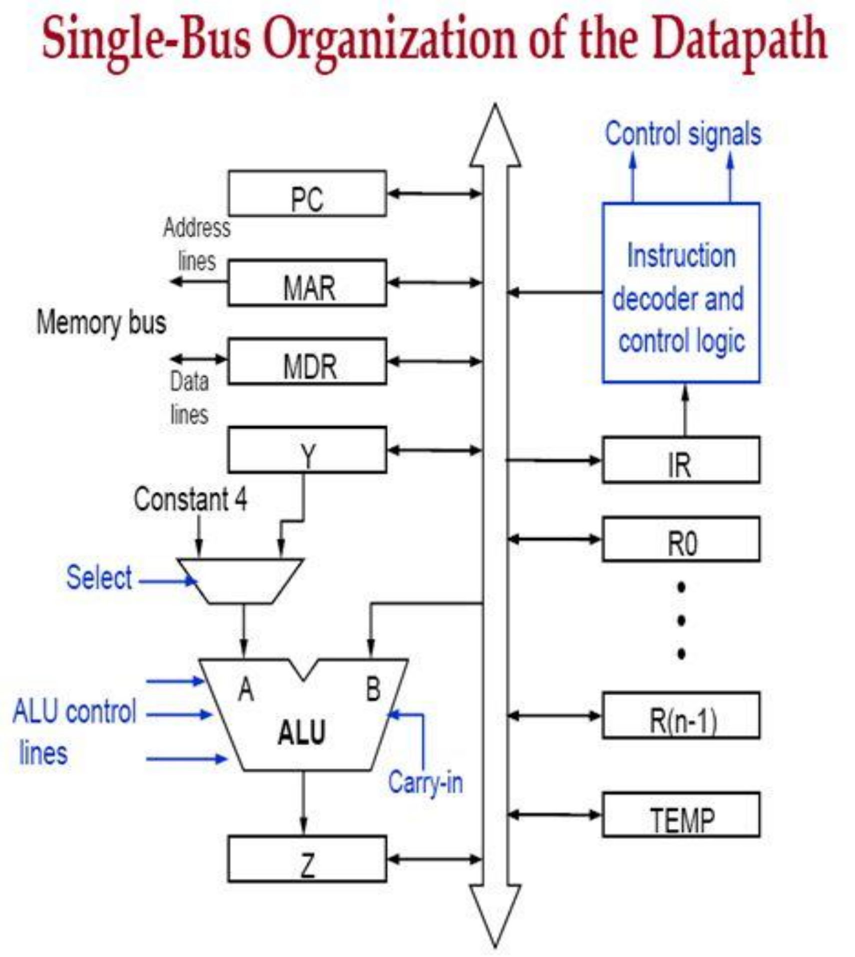
Step6: When Read operation is completed, memory-operand is available in MDR, and the addition is performed.

Step7: Sum is stored in Z, then transferred to R1.

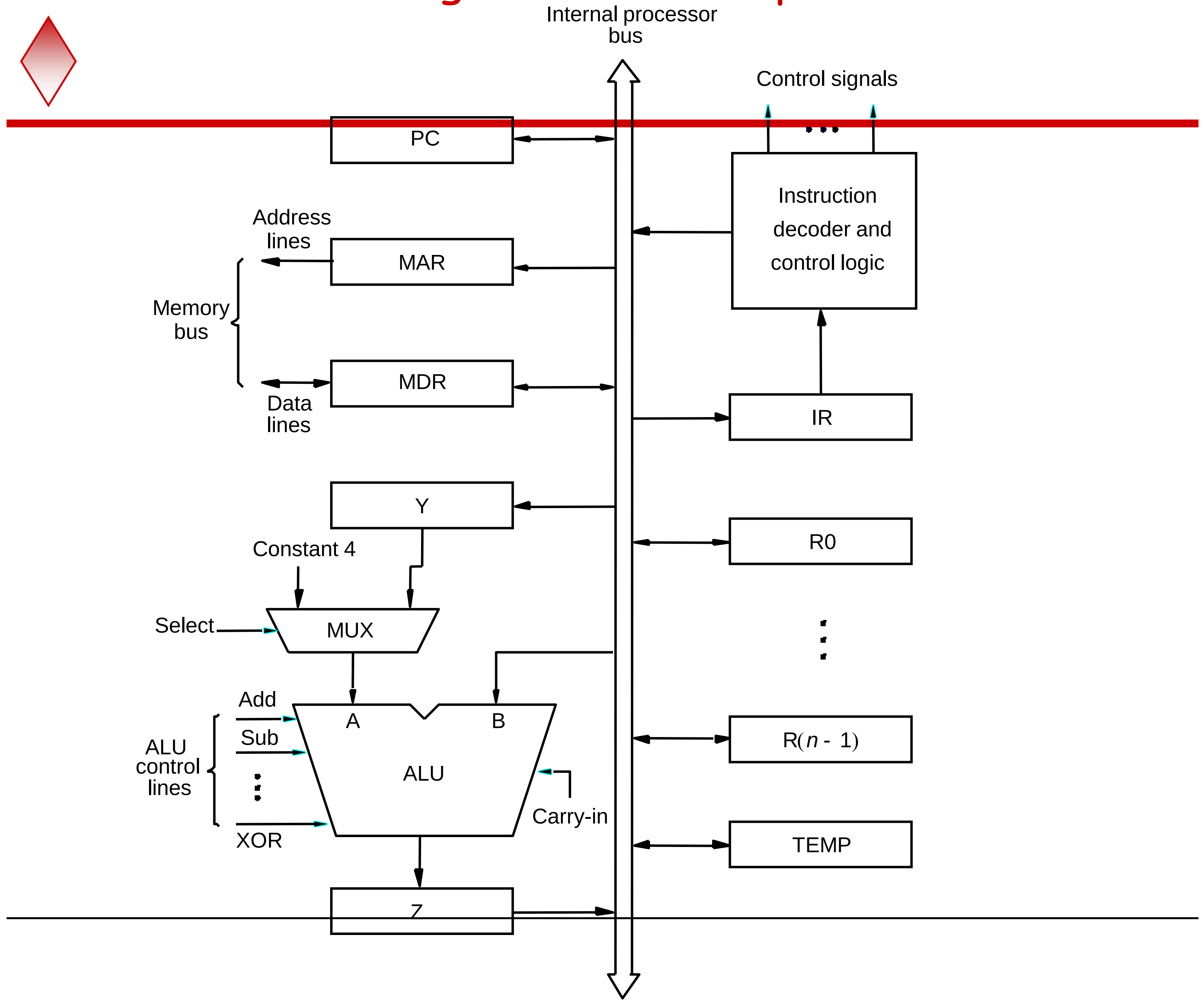
The End signal causes a new instruction fetch cycle to begin by returning to step1.

Single Bus organization of the data path within a CPU

- The control unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus.
- The MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU.
- The constant 4 is used to increment the contents of the program counter.



Internal organization of a processor



Execution of a Complete Instruction

Add (R3), R1 R1$\leftarrow$ R1 + [R3])

Execution of a Complete Instruction

Add R2, R1

$R2_{out}$

Execution of Branch Instructions

- A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction.
- The offset X is usually the difference between the branch target address and the address immediately following the branch instruction.
- Unconditional branch

Execution of Branch Instructions

The processing starts, as usual, the fetch phase ends in step3.
In step 4, the offset-value is extracted from IR by instruction-decoding circuit.

Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.

In step 5, the result, which is the branch-address, is loaded into the PC.

The offset X used in a branch instruction is usually the difference between the branch target-address and the address immediately following the branch instruction. (For example, if the branch instruction is at location 1000 and branch target-address is 1200, then the value of X must be 196, since the PC will be containing the address 1004 after fetching the instruction at location 1000).

In case of conditional branch, we need to check the status of the condition-codes before loading a new value into the PC.

e.g. Offset-field-of-IRout, Add, Zin, If N=0 then END

If N=0, processor returns to step 1 immediately after step 4.

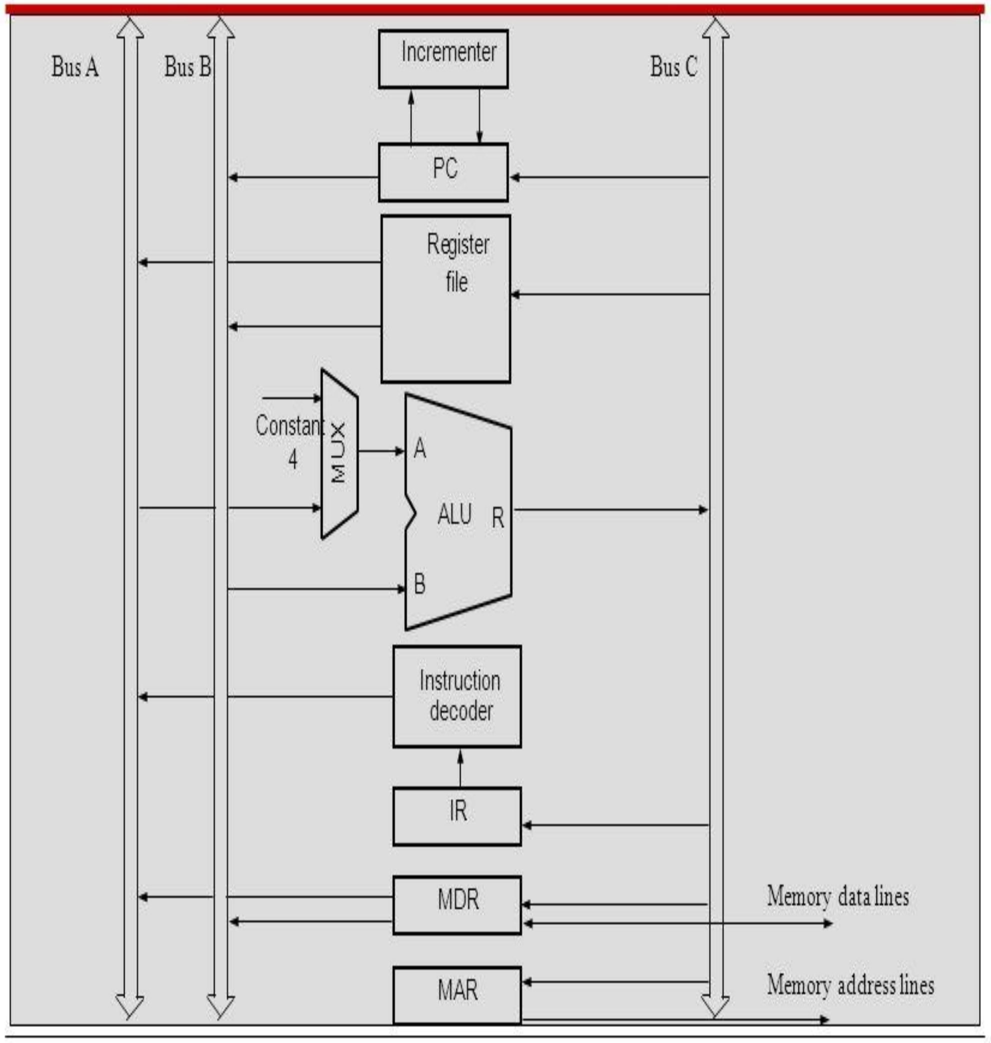
If N=1, step 5 is performed to load a new value into PC

Execution of Branch Instructions

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
2	Z_{out}, PC_{in} , Y_{in}, WMF C
3	MDR_{out} , IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in} , End

Fig. Control sequence for an unconditional branch instruction.

Multiple bus organization (contd..)



- Allow the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B.
- Allow the data on bus C to be loaded into a third register during the same clock cycle.
- Incrementer unit.
- ALU simply passes one of its two input operands unmodified to bus C

→ control signal: R=A or R=B

- General purpose registers are combined into a single block called registers.
- 3 ports,2 output ports –access two different registers and have their contents on buses A and B
- Third port allows data on bus c during same clock cycle.
- Bus A & B are used to transfer the source operands to A & B inputs of the ALU.
- ALU operation is performed.
- The result is transferred to the destination over the bus C.
- ALU may simply pass one of its 2 input operands unmodified to bus C.
- The ALU control signals for such an operation R=A or R=B.
- Incrementer unit is used to increment the PC by 4.
- Using the incrementer eliminates the need to add the constant value 4 to the PC using the main ALU.
- The source for the constant 4 at the ALU input multiplexer can be used to increment other address such as loadmultiple & storemultiple

Multiple-Bus Organization

- Add R4, R5, R6   for the three-bus organization

Step	Action
1	PC_{out}, R=B, MAR_{in} , Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Fig. Control sequence for the instruction

.

Instruction execution proceeds as follows in Multiple-Bus Organization

Step 1:The contents of PC are passed

through the ALU using R=B control signal & loaded into MAR to start a memory read operation

At the same time PC is incrementer by 4

Step 2:The processor waits for MFC

Step 3: Loads the data ,received into MDR ,then transfers them to IR.

Step 4: The execution phase of the instruction requires only one control step to complete.

Exercise

- What is the control sequence for execution of the instruction

  Add R1, (R2)

  including the instruction fetch phase? (Assume single bus architecture)

The Processing Unit

1. Basic Processing Cycle
2. Types of Operations
3. **Control Mechanisms**

↑

Q Who sets F_alu to *ADD*?

1. Hardwired
2. Micro-Programmed

87

Hardwired Control

3 Control Mechanism- Overview

- To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence.
- Two categories: hardwired control and micro programmed control
- Hardwired system can operate at high speed; but with little flexibility.

Hardwired vs Microprogrammed

- Hardwired
  - Use gates to generate signals
  - Squeeze out the juice for performance(not flexible)
  - Different logic styles possible
  - Economical initially
  - Small change→redesign
- Microprogrammed
  - Store the control signals in the sequence
  - Just read from the memory every clock cycle
  - Expensive initially
  - Additions done by simply changing the microprogram in control memory
  - Diagnostics routine can be made available in memory

3.1 HARDWIRED APPROACH

- Final circuit is obtained by physically connecting gates and flip flops
- Cost of control logic increases with system complexity

Control Unit Organization

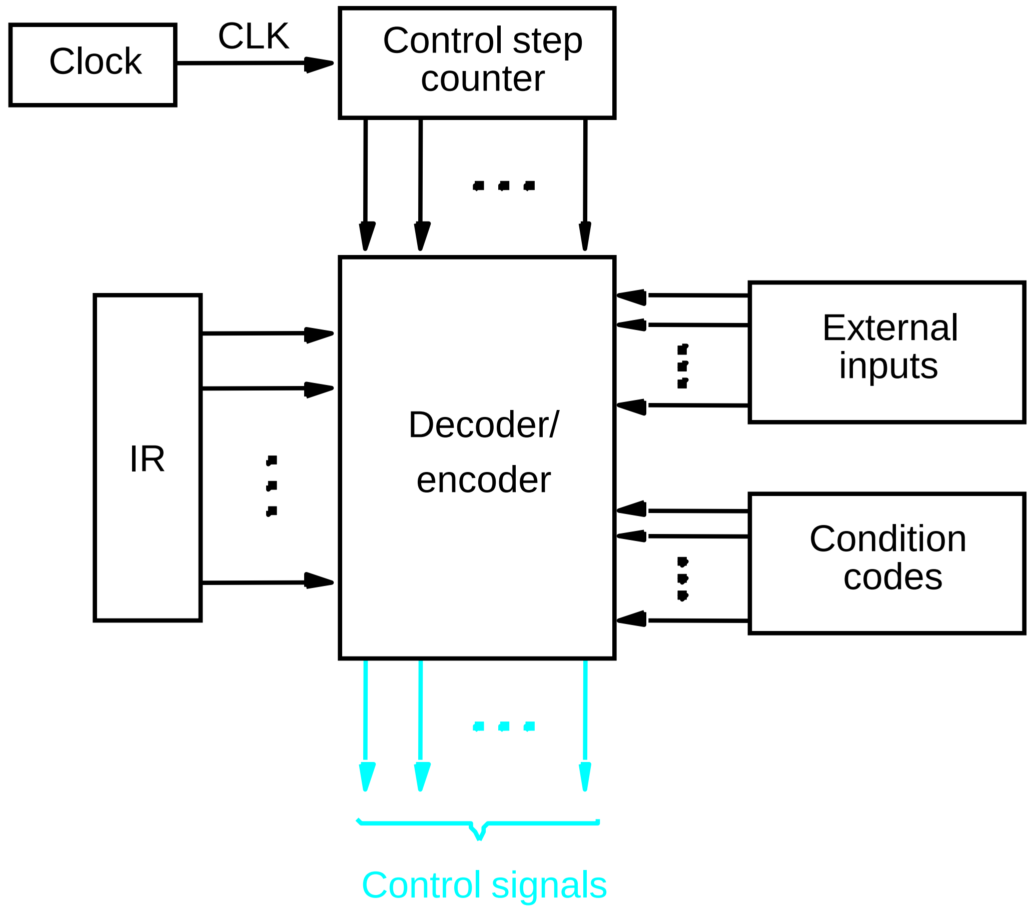
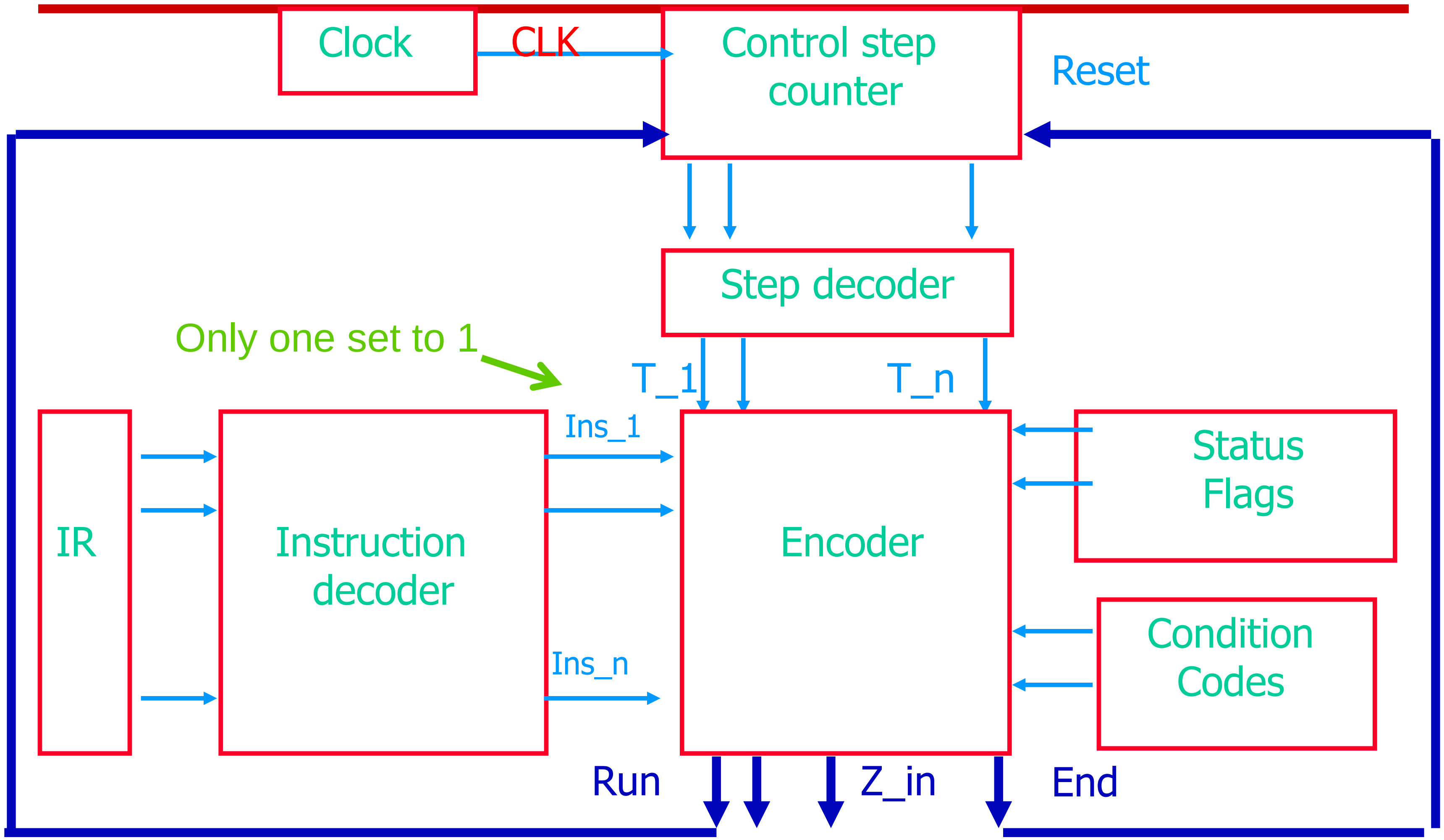


Fig.Control unit organization.

Hardwired Control -Separating decoding/encoding



Q) Role of Run ?

Decoder/encoder block is a combinational-circuit that generates required control-outputs depending on state of all its inputs.

Step-decoder provides a separate signal line for each step in the control sequence.

Similarly, output of instruction-decoder consists of a separate line for each machine instruction.

For any instruction loaded in IR, one of the output-lines INS1

through INSm is set to 1, And all other lines are set to

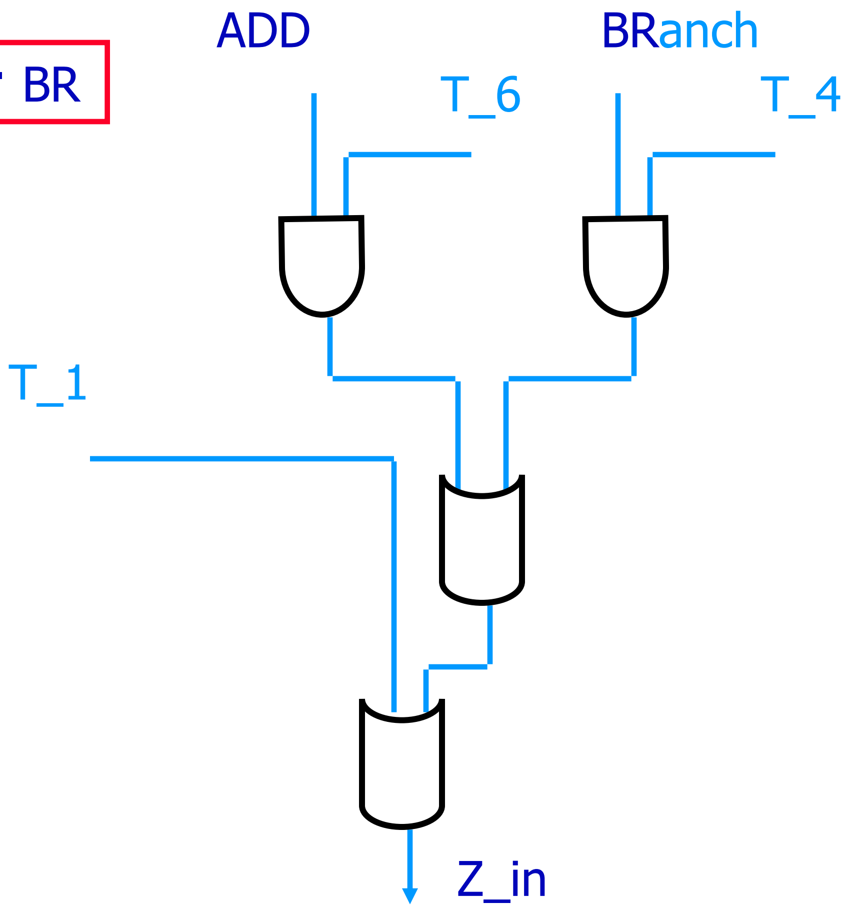
The input signals to encoder-block are combined to generate the individual control- signals Yin, PCout, Add, End and so on.

3.1. Hardwired Control -Generation of control signals

$\boxed{Z\_in = T\_1 + T\_6 \cdot ADD + T\_4 \cdot BR}$

↑ ↑ ↑

time slot



This signal is asserted during time-slot T1 for all instructions,

during T6 for an Add instruction

during T4 for unconditional branch instruction

The control H/W shown can be viewed as a state m/c that changes from one state to another in every clock cycle, depending on the content of IR,the conditional codes and the external inputs

When RUN=1, counter is incremented by 1 at the end of every clock cycle.

When RUN=0, counter stops counting.

Sequence of operations carried out by this machine is determined by wiring of logic elements, hence the name “hardwired”.

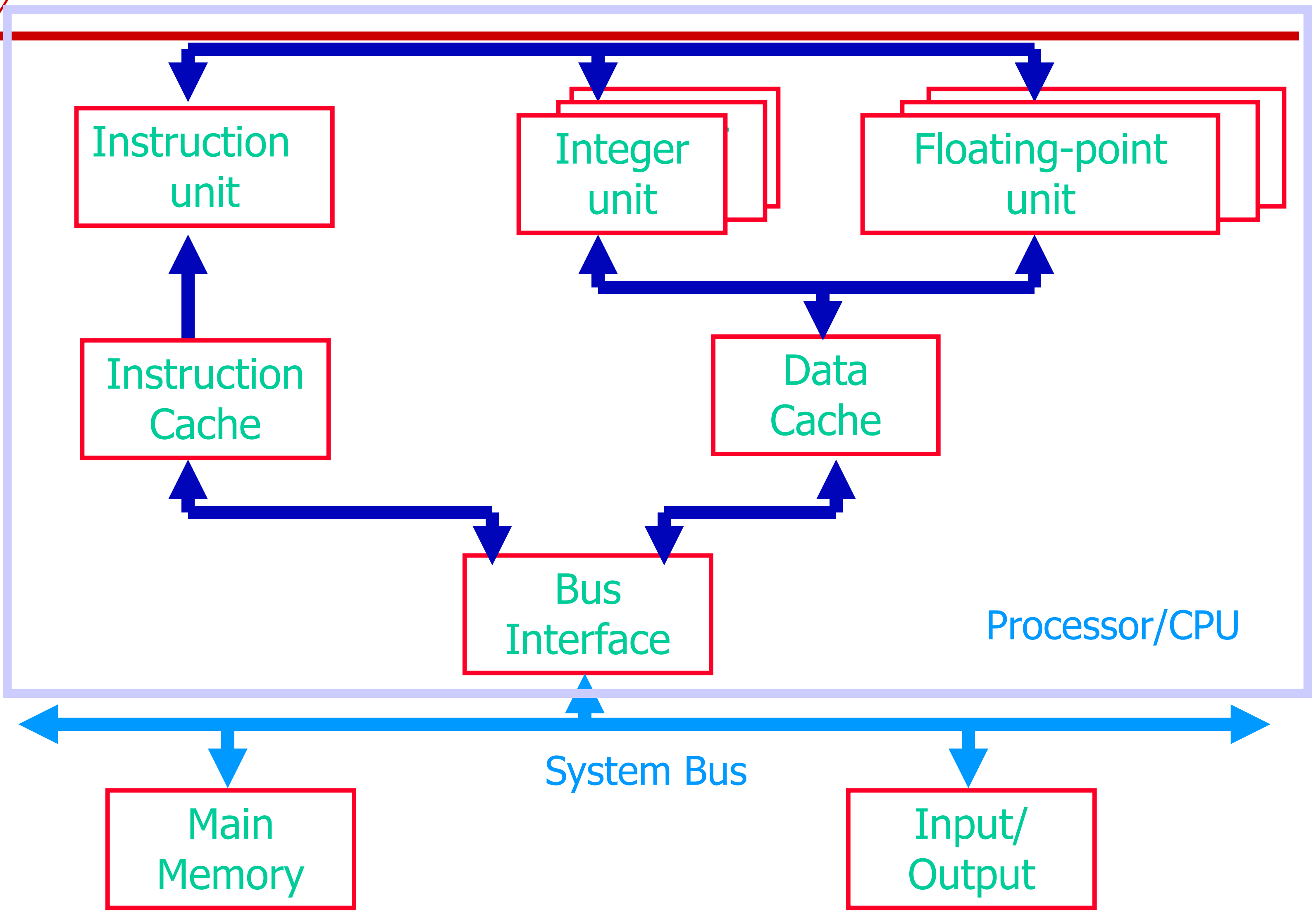
Generation End control signal

End = T_7 · ADD + T_5 · BR +(T_5 · N + T_4 · /N) ·BRN+ . . . .

3.1. Hardwired Control-Performance

- Performance is dependent on:
  - Power of instructions
  - Cycle time
  - Number of cycles per instruction
- Performance improvement by:
  - Multiple datapaths
  - Instruction prefetching and pipelining
  - Caches

Complete CPU



A Complete Processor

- ❖This has separate processing-units to deal with integer data and
- ❖ floating-point data.
- ❖ A data-cache is inserted between these processing-units & main-memory.
- ❖Instruction-unit fetches instructions
  - → from an instruction-cache or
  - → from main-memory when desired instructions are not already in cache
- ❖ Processor is connected to system-bus & hence to the rest of the computer by means of a bus interface
- ❖ Using separate caches for instructions & data is common practice in many processors today.
- ❖ A processor may include several units of each type to increase the potential for concurrent operations

Micro programmed Control

3.2. Micro-programmed control

- All control bits are organized as memory
- Each memory location represents a control setting/word
  - Control word (CW) consists of individual bits represent various control-signals (like Add, End, Zin).
  - The control word represents unique combination of (0s and 1s)
- Memory words in micro routine are called micro-instructions.
- Micro-routines are sequences of micro-instructions
  - Control stores for all micro-routines of the instruction set .
  - Micro-program counter (μ PC) to read control words sequentially

    Every time a new instruction is loaded into IR, output of "starting address generator" is loaded into μPC.

    Then, μPC is automatically incremented by clock, causing successive microinstructions to be read from CS.
- Control-signals are generated by a program similar to machine language programs

3.2. Micro-Programmed Control
Examples of Micro-Instructions

Micro-instruction ...	PC_in	MAR_in	Z_in
1 ...	0	1	1
2 ...	1	0	0
3 ...	0	0	0.

Microprogrammed Control

- Control signals are generated by a program similar to machine language programs.
- Control Word (CW); microroutine; microinstruction

Microinstructions for Add (R3),R1

Microroutine for the instruction Branch<0

- The previous organization cannot handle the situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action.
- Use conditional branch microinstruction.

Address	Microinstruction
0	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
1	Z_{out} , PC_{in} , Y_{in} , WMFC
2	MDR_{out} , IR_{in}
3	Branch to starting address of appropriate microroutine
. . . .	
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out} , SelectY, Add, Z_{in}
27	Z_{out} , PC_{in} , End

Fig. Microroutine for the instruction Branch<0.

Micro programme Unit for Conditional Branching

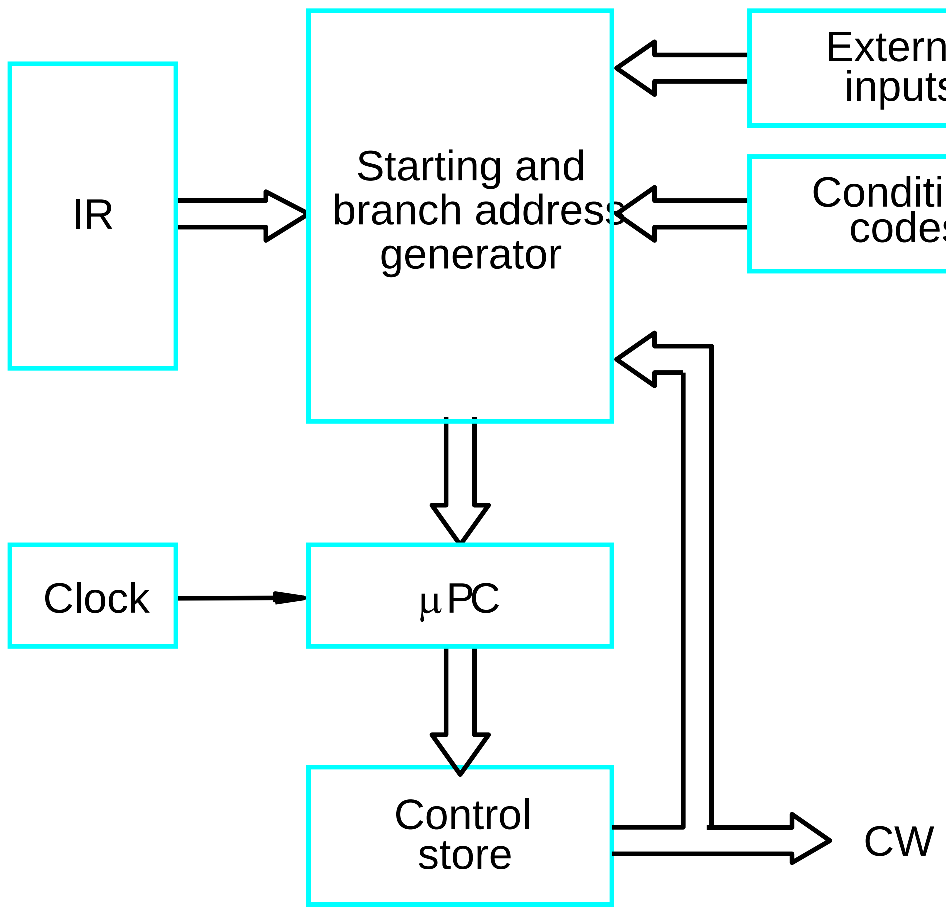


Fig. Organization of the control unit to allow conditional branching in the micro program.

Conditional Branching

In case of conditional branching, microinstructions specify which of the external inputs, condition-codes should be checked as a condition for branching to take place.

The *starting and branch address generator block* loads a new address into μPC when a microinstruction instructs it to do so .

To allow implementation of a conditional branch, inputs to this block consist of
→ external inputs and condition-codes
→ contents of IR

μPC is incremented every time a new microinstruction is fetched from microprogram memory except in following situations

i) When a new instruction is loaded into IR, μPC is loaded with starting-address of microroutine for that instruction.

ii) When a Branch microinstruction is encountered and branch condition is satisfied, μPC is loaded with branch-address.

iii) When an End microinstruction is encountered, μPC is loaded with address of first CW in microroutine for instruction fetch cycle.

Microinstructions

- A straightforward way to structure microinstructions is to assign one bit position to each control signal.
- However, this is very inefficient.
- The length can be reduced: most signals are not needed simultaneously, and many signals are mutually exclusive.
- All mutually exclusive signals are placed in the same group in binary coding.

Some Drawbacks of microprogrammed control

1) Assigning individual bits to each control-signal results in long microinstructions because the number of required signals is usually large.
2) Available bit-space is poorly used because

- only a few bits are set to 1 in any given microinstruction.

Solution: Signals can be grouped because

- Most signals are not needed simultaneously.
- Many signals are mutually exclusive.

Grouping control-signals into fields requires a little more hardware because decoding-circuits must be used to decode bit patterns of each field into individual control signals.

Advantage: This method results in a smaller control-store (only 20 bits are needed to store the patterns for the 42 signals).

3) Having a separate microroutine for each machine instruction results in a large total number of microinstructions and a large control-store.
4) Execution time is longer because it takes more time to carry out the required branches.

Microprogrammed control (contd..)

Microinstruction format

- Simple approach is to allocate one bit for each control signal
  - Results in long microinstructions, since the number of control signals is usually very large.
  - Few bits are set to 1 in any microinstruction, resulting in a poor use of bit space.
- Reduce the length of the microinstruction by taking advantage of the fact that most signals are not needed simultaneously, and many signals are mutually exclusive. For example:
  - Only one ALU function is active at a time.
  - Source for a data transfer must be unique.
  - *Read* and *Write* memory signals cannot be active simultaneously.

- Group mutually exclusive signals in the same group.
- At most one microoperation can be specified per group.
- Use binary coding scheme to represent signals within a group.

Microprogrammed control (contd..)

Microinstruction format

Examples:

- If ALU has 16 operations, then 4 bits can be sufficient.
- Group register output signals into the same group, since only one of these signals will be active at any given time (Why?)

If the CPU has 4 general purpose registers, then PC_{out}, MDR_{out}, Z_{out}, $Offset_{out}$, $R0_{out}$, $R1_{out}$, $R2_{out}$, $R3_{out}$ and $Temp_{out}$ can be placed in a single group, and 4 bits will be needed to represent these.

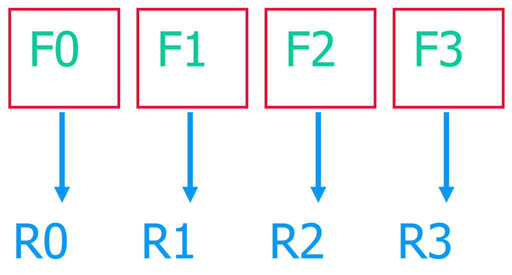
3.2. Micro-Programmed Control
Structure micro-instructions

- ❑ Most simple organization: 1 bit per control signal
- ❑ However,
  - ◆ Many bits needed (e.g., 80-120 bits)
  - ◆ For many signals, only one is needed per cycle; hence they can be grouped
  - ◆ Coding is possible: e.g., an address instead of a single control bit per register
- ❑ Alternative approach to reduce the length of the microinstruction:
- • Enumerate the patterns of the required signals in all microinstructions.
- • Assign each meaningful combination of active control signals a unique code.
- • Code represents the microinstruction .
- • Grouping control signals into fields requires a little more hardware:
- ❑ Decoding circuits are needed to decode patterns to individual control signals.
- ❑ Cost of the additional hardware is offset by reduced number of bits
- ❑ in each microinstruction:
- ❑ Reduces the size of the control store.
- ❑ Full encoding can reduce the length of the microinstruction even further, but this reduction comes at the expense of increasing the complexity of the decoder circuits.

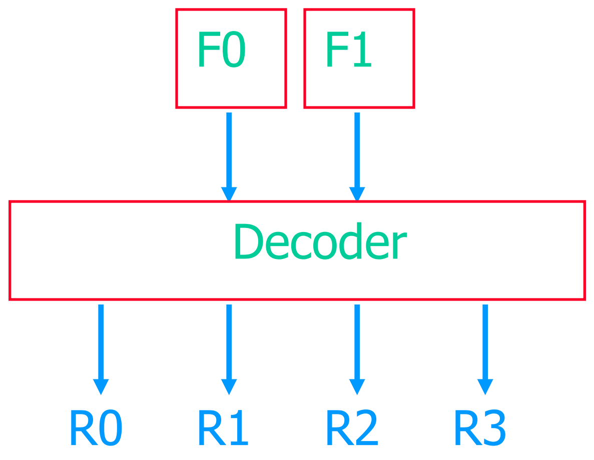
3.2. Micro-Programmed Control
Forms of organization

- Little coding: horizontal organization
  - Large words
  - Little decoding logic
  - Fast
- Much coding: vertical organization
  - Small control store
  - Much decoding logic
  - Slower
- Mixed organization

3.2. Micro-Programmed Control
Horizontal/Vertical



Horizontal



Vertical

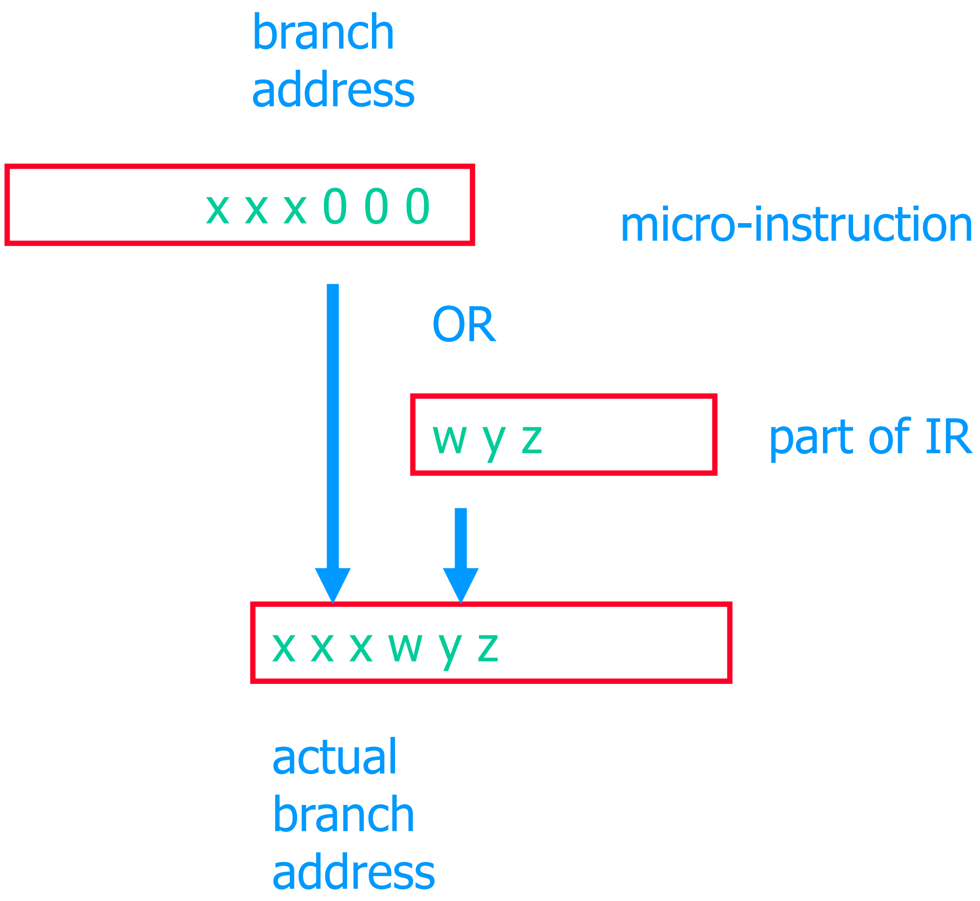
3.2. Micro-Programmed Control
Sequencing

- Thus far only branch after fetch
- No sharing of micro-code between micro-routines
- Micro-subroutines lead to more efficient control store

3.2. Micro-Programmed Control
Multi-way branching

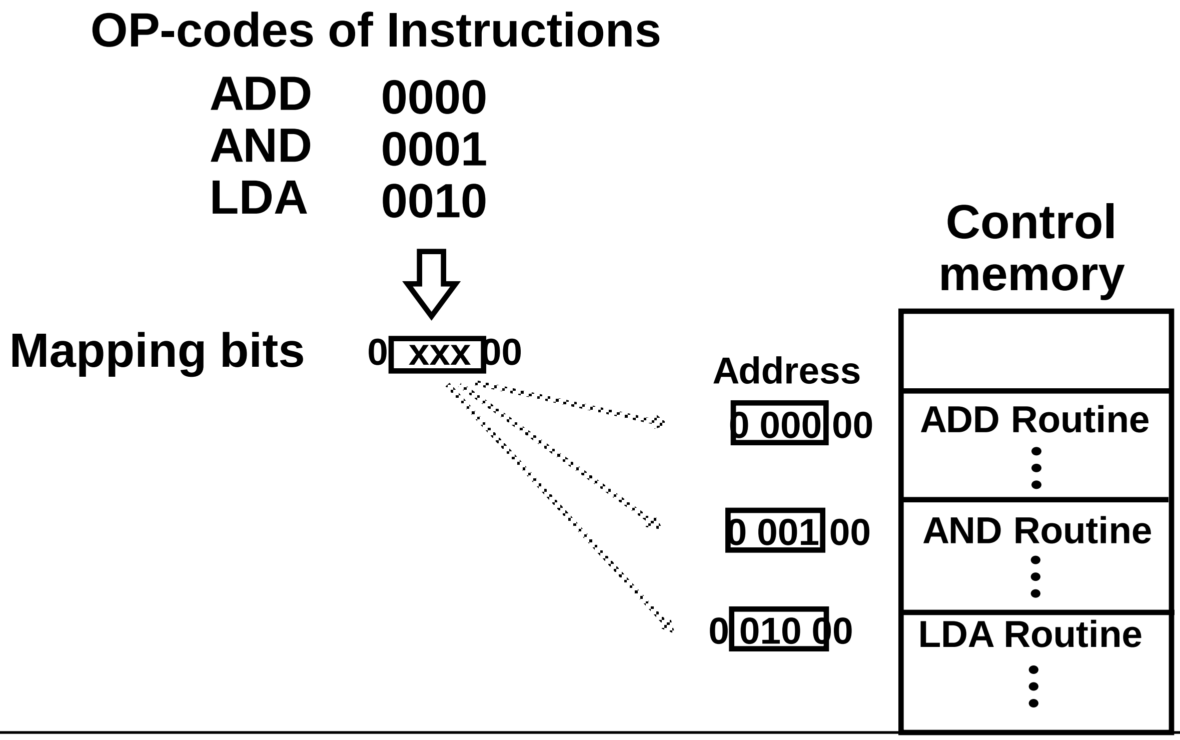
- Number of two-way branches
  - disadvantage: slows down
- More than one branch address in micro-instruction
  - disadvantage: more bits required
- bit-ORing if specified branch address

3.2. Micro-Programmed Control
Example



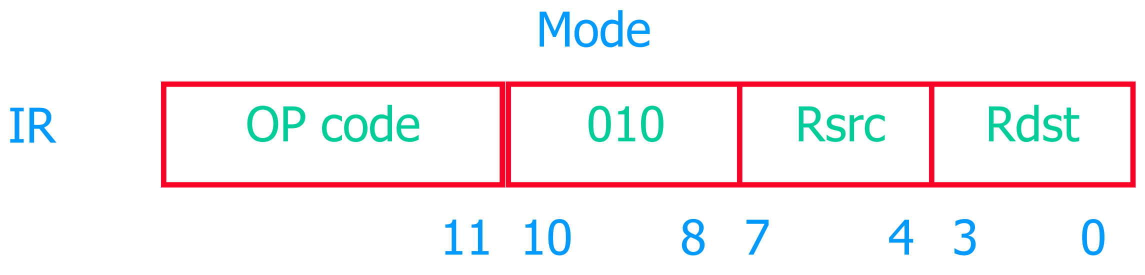
Mapping of Instruction

- Example
  - Mapping 3-bit operation code to 7-bit address



3.2. Micro-Programmed Control
Example microroutine (1)

ADD (Rsrc)+, Rdst



Instruction Format

bit 8: direct/indirect
bit 9,10: indexed (11)
autodecrement(10)
autoincrement(01)
register(00)

3.2. Micro-Programmed Control
Example microroutine (2)

Address	Micro-instruction
0	PC_out, MAR_in, Read, Set carry-in ALU, F_alu = "ADD", Z_in
·	Z_out, PC_in, Wait for MFC
2	MDR_out, IR_in
3	μBranch{μPC←101 (from PLA); μPC_5,4←[IR_10,9]; μPC_3←{[not.IR_10].[not.IR_9].[IR_8]}

FETCH

use bits from IR for addressing mode

....

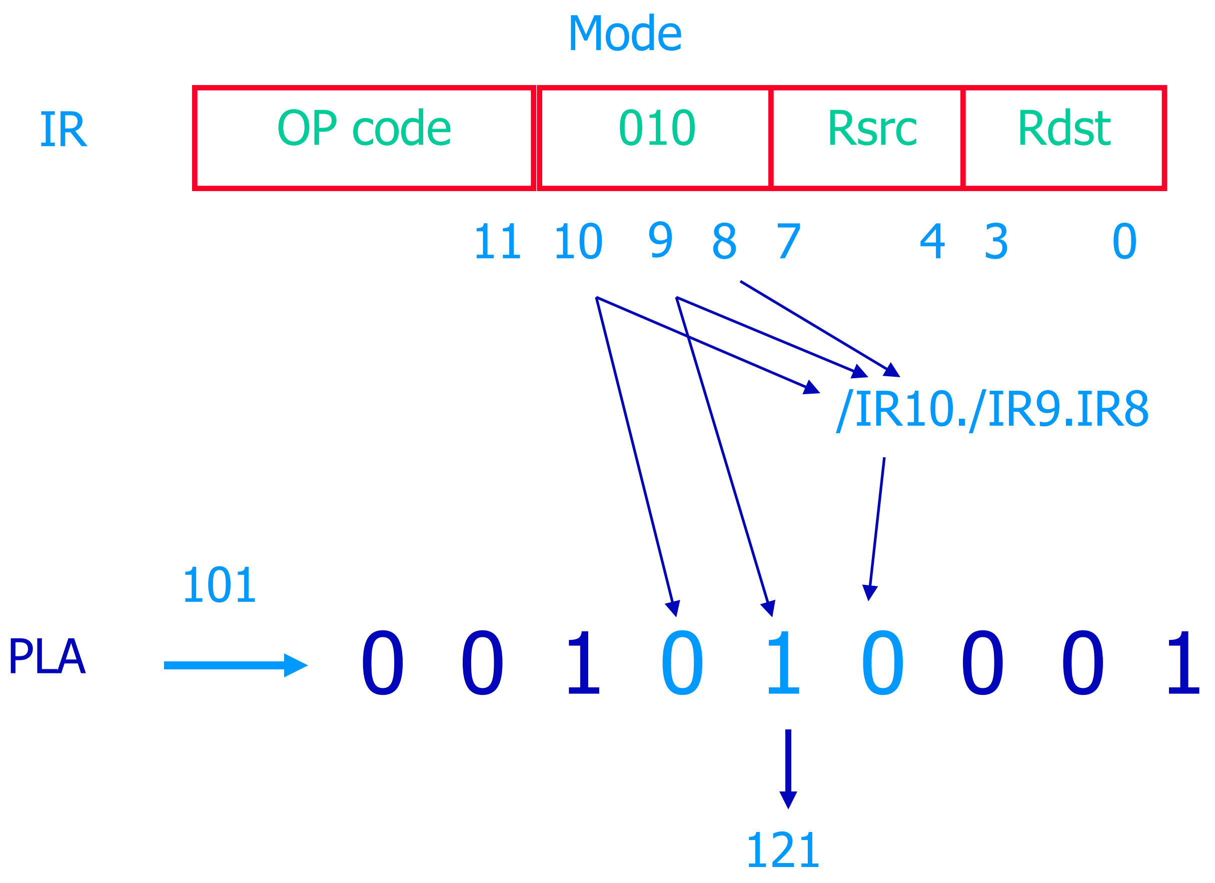
Address	Micro-instruction
121	Rsrc_out, MAR_in, Set carry-in ALU,Read, F_alu = "ADD", Z_in
122	Z_out, Rscr_in
123	μBranch{μPC←170; μPC_0←[not.IR_8]}, WMFC
170	MDR_out, MAR_in, Read, WMFC
171	MDR_out, Y_in
172	Rdst_out, F_alu = "ADD", Z_in
173	Z_out, Rdst_in, End

autoincrement

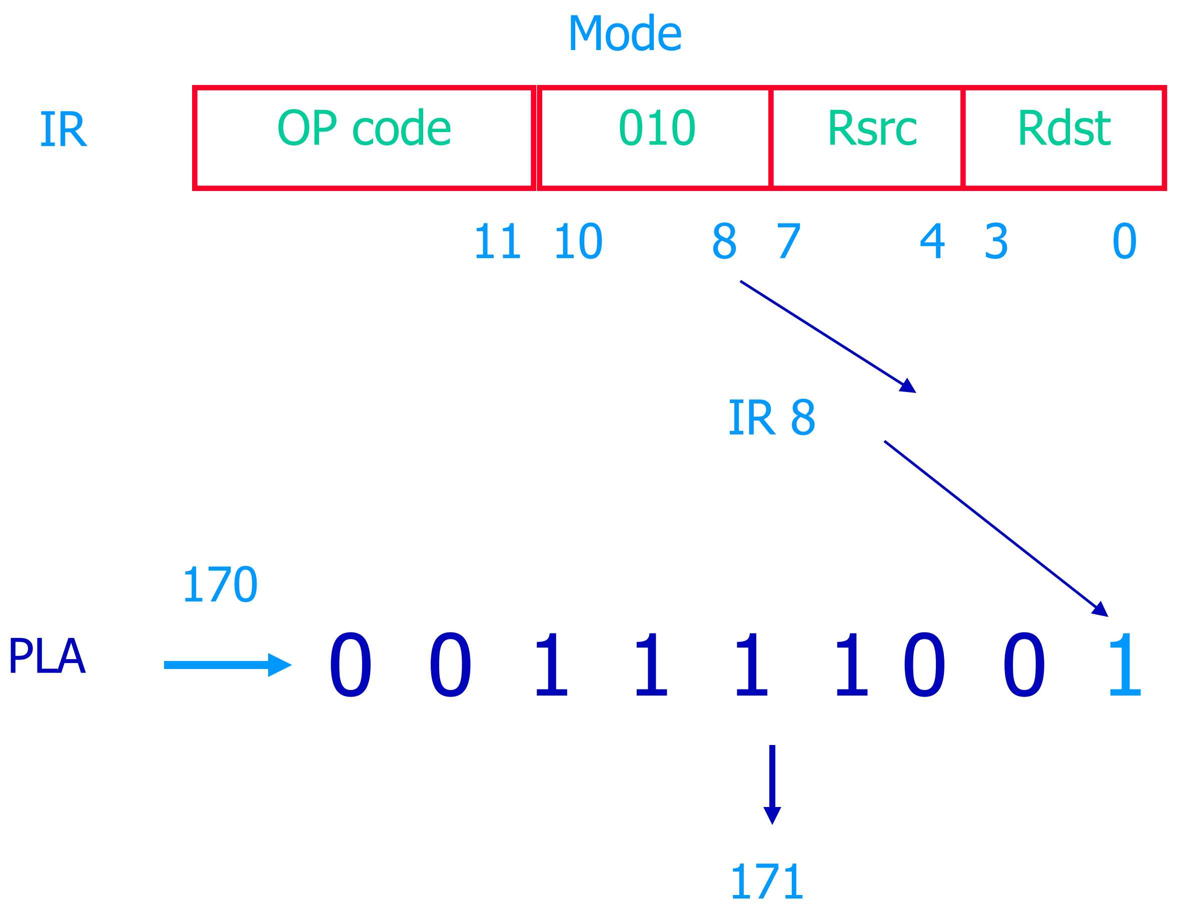
indirect

direct

3.2. Micro-Programmed Control
Micro branch address



3.2. Micro-Programmed Control
Micro branch address



Microprogram sequencing-

Simple microprogram sequencing:

•Load the starting address into μPC when a new instruction is loaded into IR.

•Introduce some branching capability within the microprogram through special branch microinstructions, which specify the branch address.

Disadvantages of the simple approach:

•Large total number of microinstructions and large control store.

•Most machines have several addressing modes, and many combinations of instructions and addressing modes.

•Separate microroutine for each of these combinations produces a lot of

•duplication of common parts.

•Share as much common code as possible.

•Sharing common code requires many branch instructions to transfer control among various parts.

•Execution time is longer because it takes more time to carry out the required branches.

Microprogrammed control (contd..)

Microprogram sequencing

Consider the following instruction which adds the source operand to the contents of register Rdst, and places the results in register Rdst.

ADD src, Rdst

Assume that the source operand can be specified in the following addressing modes:

- Register.
- Autoincrement
- Autodecrement
- Indexed.
- Indirect forms of the above methods.

ADD Rsrc, Rdst

Start of instruction fetch.

Microroutines for all the instructions

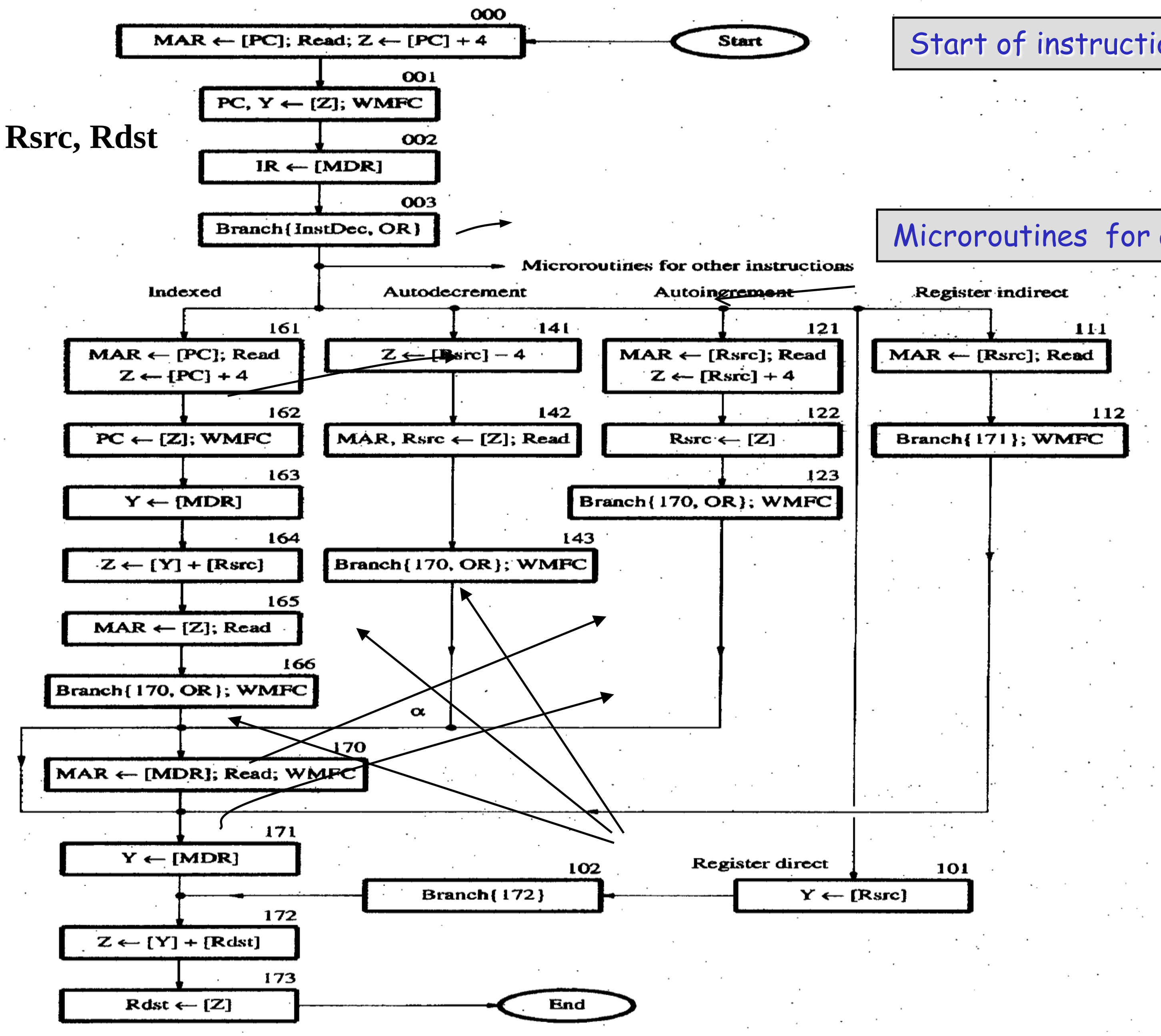


Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

Microprogrammed control (contd..)

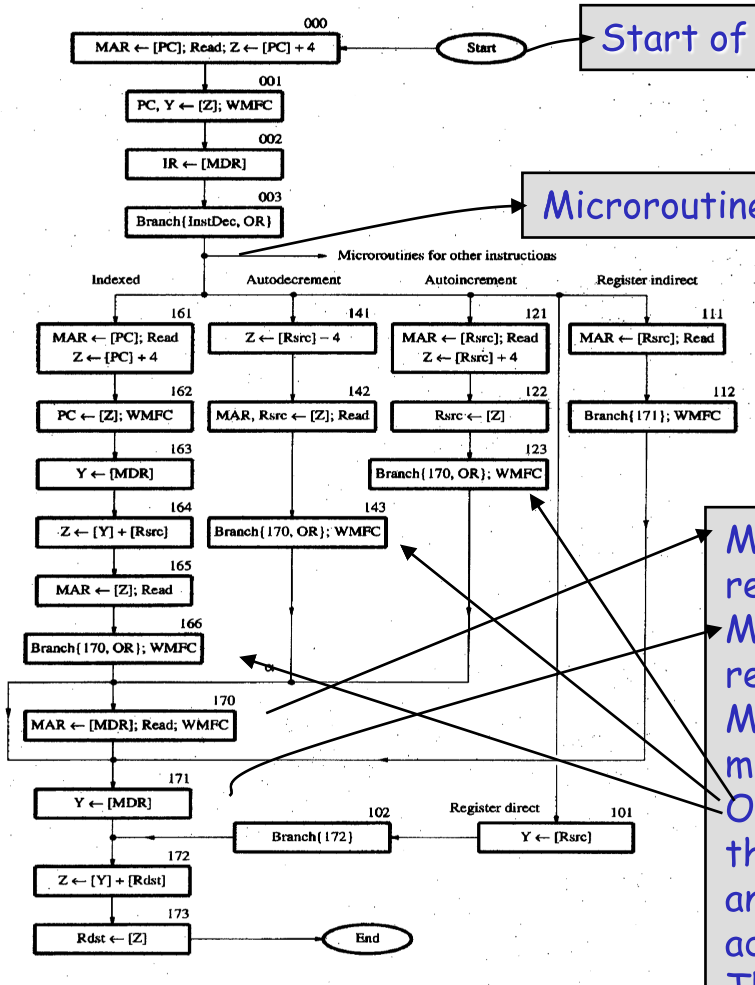


Figure 7.20. Flowchart of a microprogram for the Add src,Rdst instruction.

Start of instruction fetch.

ADD Rsrc, Rdst

Microroutines for all the instructions

Microinstruction at address 170 is performed in the register indirect mode.
Microinstruction at address 171 is performed in the register direct mode.
Microinstruction 170 is bypassed if register direct mode is used.
One way to bypass microinstruction 170 is to have the previous microinstruction specify address 170, and then use an OR gate to change the LSB of the address to 1.
This technique is called bit-ORing.

Branch Address Modification Using Bit-ORing

The micro program in Figure 20 shows that branches are not always made to a single branch address. This is a direct consequence of combining simple micro routines by sharing common parts. Consider the point labeled ∞ in the figure. At this point, it is
necessary to choose between actions required by direct and indirect addressing modes.
If the indirect mode is specified in the instruction, then the microinstruction in location 170 is performed to fetch the operand from the memory. If the direct mode is specified, this fetch must be bypassed by branching immediately to location 171.
The most efficient way to bypass microinstruction 170 is to have the preceding branch microinstructions specify the address 170 and then use an OR gate to change the least significant it of this address to 1 if the direct addressing mode is involved.
This is known as the bit-ORing technique for modifying branch addresses.

Alternative Approaches to the bit-ORing

An alternative to the bit-ORing approach is to use two conditional branch microinstructions allocations 123,143, and 166.
Another possibility is to include two next address fields within a branch microinstruction, one for the direct and one for the indirect address modes.
Both of these alternatives are inferior to the bit-Oring technique.

Microprogrammed control (contd..)

Microinstructions with the next-address field.

- Several branch microinstructions are required to enable sharing of common code.
- The branch microinstructions do not perform any useful operation related to data.
- They are required to determine the address of the next microinstruction.
- They slow down the execution of the instruction.

- Ideally we need to assign consecutive addresses to all microinstructions that are generally executed one after the other.
- Recall that the next microinstruction is determined by incrementing the microprogram counter.
- But due to the goal of sharing as much common code as possible, this is not always possible.
- This leads to a significant increase in the branch instructions.

Microprogrammed control (contd..)

Microinstructions with the next-address field.

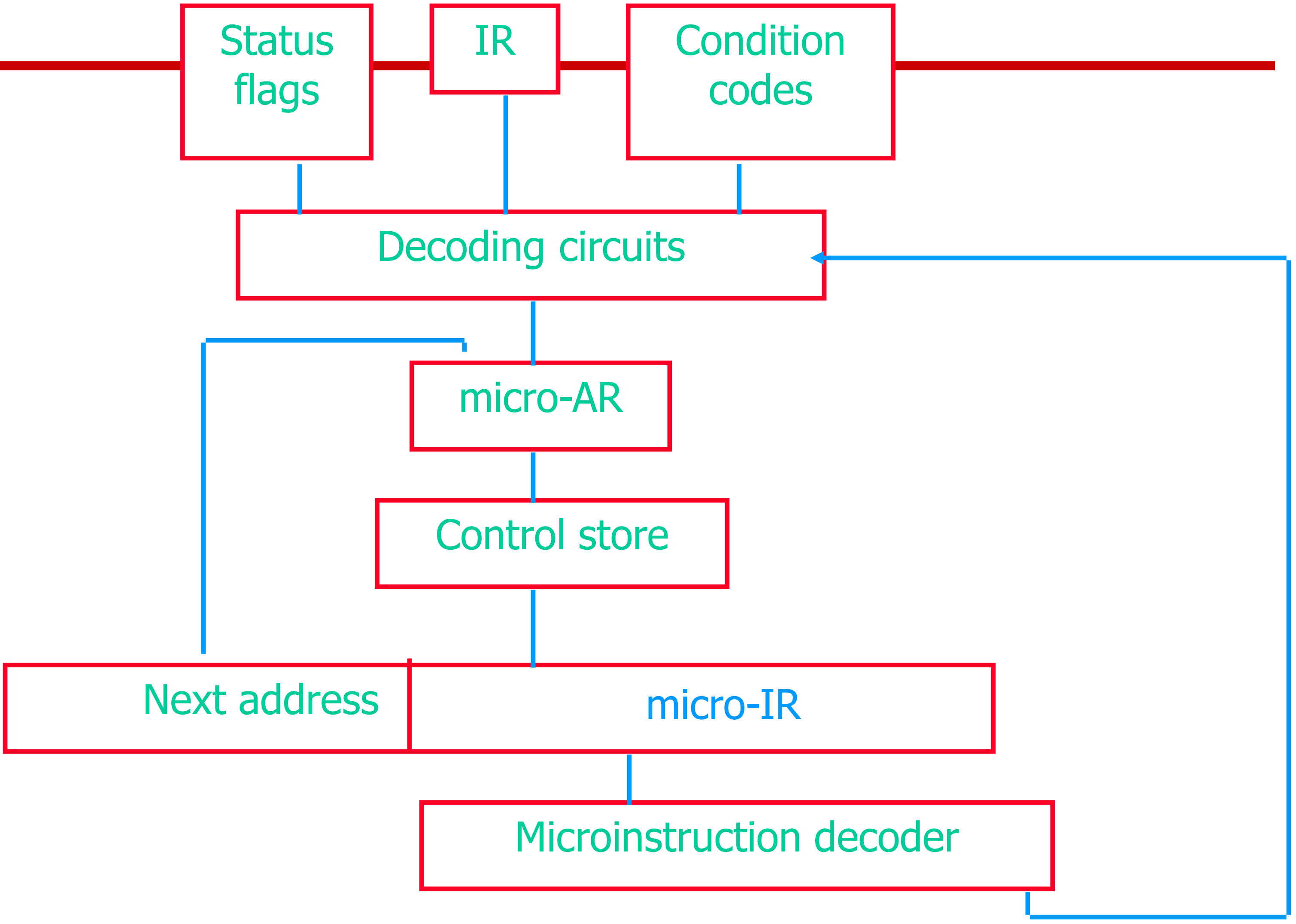
•Powerful alternative is to include an address field as a part of every microinstruction.
•The address field indicates the location of the next microinstruction to be fetched.
•In effect, every microinstruction becomes a branch microinstruction in addition to its other function.

Disadvantages:
•Additional bits are required to specify the address field in every instruction.
•Approximately one-sixth of the control store is devoted to specifying the address.
Advantages:
•Separate branch instructions are virtually eliminated.
•Flexible scheme, very few restrictions in assigning addresses to microinstructions.

3.2. Micro-Programmed Control
Next-address field (1)

- Micro-instruction contains address next micro-instruction
- Larger store needed
- Branch micro-instructions no longer needed

Next-address field (2)



Next-address field

- The microprogram discussed earlier requires several branch microinstructions, which perform no useful operation in the datapath.
- A powerful alternative approach is to include an address field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched.
- Pros: separate branch microinstructions are virtually eliminated; few limitations in assigning addresses to microinstructions.
- Cons: additional bits for the address field (around 1/6)

3.2. Micro-Programmed Control
Example

F0	F1	F2	F3	F4	F5	F6	F7	F8

Field 0(8 bits):	Next address
Field 1(4 bits):	Register address_in
Field 2(4 bits):	Register address_out
Field 3(4 bits):	Other registers_in
Field 4(4 bits):	Function ALU
Field 5(2 bit)	: Read/Write/Nop
Field 6(1 bit) :	Carry-in ALU
Field 7(1 bit) :	WMFC
Field 8(1 bit) :	End
....	PLA/ORing etc

Partial Format for the field encoded Microinstructions

Microinstruction

F1	F2	F3	F4	F5

F1 (4 bits)	F2 (3 bits)	F3 (3 bits)	F4 (4 bits)	F5 (2 bits)
0000: No transfer	000: No transfer	000: No transfer	0000: Add	00: No action
0001: PC_{out}	001: PC_{in}	001: MAR_{in}	0001: Sub	01: Read
0010: MDR_{out}	010: IR_{in}	010: MDR_{in}	⋮	10: Write
0011: Z_{out}	011: Z_{in}	011: $TEMP_{in}$		
0100: $R0_{out}$	100: $R0_{in}$	100: Y_{in}	1111: XOR	
0101: $R1_{out}$	101: $R1_{in}$		16 ALU functions	
0110: $R2_{out}$	110: $R2_{in}$			
0111: $R3_{out}$	111: $R3_{in}$			
1010: $TEMP_{out}$				
1011: $Offset_{out}$				

F6	F7	F8	...

F6 (1 bit)	F7 (1 bit)	F8 (1 bit)
0: SelectY	0: No action	0: Continue
1: Select4	1: WMFC	1: End

What is the price paid for this scheme?

- *Each group occupies a large enough field to represent all the signals.*
- *Most fields must include one inactive code, which specifies no action.*
- *All fields do not have to include inactive code.*

Require a little more hardware

3.2. Micro-Programmed Control Organization

- Micro-program is often placed in ROM on CPU chip
- Some machines had writable control store, i.e. user could change instruction set

Further Discussions

- Prefetching
- Emulation

Prefetching

- Microprogrammed control leads to slower operating speed because of the time it takes to fetch microinstructions from the control store.
- To achieve faster operation, the next microinstruction can be prefetched while the current one is being executed.
  - Execution time can be overlapped with the fetch time.
  - the next microinstruction is pre-fetched while the current one is being executed.
- Prefetching microinstructions presents some difficulties:
  - Status flags and the results of the current microinstruction that is being executed are necessary to determine the address of the next microinstruction.
  - Straightforward prefetching may occassionally fetch a wrong instruction.
  - Fetch must be repeated.
- Disadvantages/difficulties are more than balance the increased operation speed.

Emulation

- The main function of microprogrammed control is to provide a means for simple, flexible and relatively inexpensive execution of machine instruction.
- Control offers the flexibility to add new instructions to the instruction set of a processor.
  - New microroutines need to be added to implement the new instructions.
- Add to the instruction set of a given computer M_1 an entirely new set of instructions that is in fact the instruction set of a different computer M_2.
  - Programs written in the machine language of M_2 can be run on M_1.
  - M_1 emulates M_2.
- Emulation allows transition to new computer systems with minimal disruption.

❖ Emulation is easiest when the machines involved have similar architecture .

❖ Emulation allows us to replace obsolete equipment with more up-to-date

❖ machines.

❖ If the replacement computer fully emulates the original one, then ~~no software changes have to be made to run existing programs~~

Implementation of the Microroutine

Address (octal)	Microinstruction
000	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
001	Z_{out}, PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μ Branch { $\mu PC \leftarrow$ 101 (from Instruction decoder); $\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$}
121	$Rsrc_{out}$, MAR_{in}, Read, Select4, Add, Z_{in}
122	Z_{out}, $Rsrc_{in}$
123	μ Branch { $\mu PC \leftarrow$ 170; $\mu PC_0 \leftarrow [\overline{IR_8}]$}, WMFC
170	MDR_{out}, MAR_{in}, Read, WMFC
171	MDR_{out}, Y_{in}
172	$Rdst_{out}$, SelectY, Add, Z_{in}
173	Z_{out}, $Rdst_{in}$, End

Implementation of the Microroutine

bit-ORing

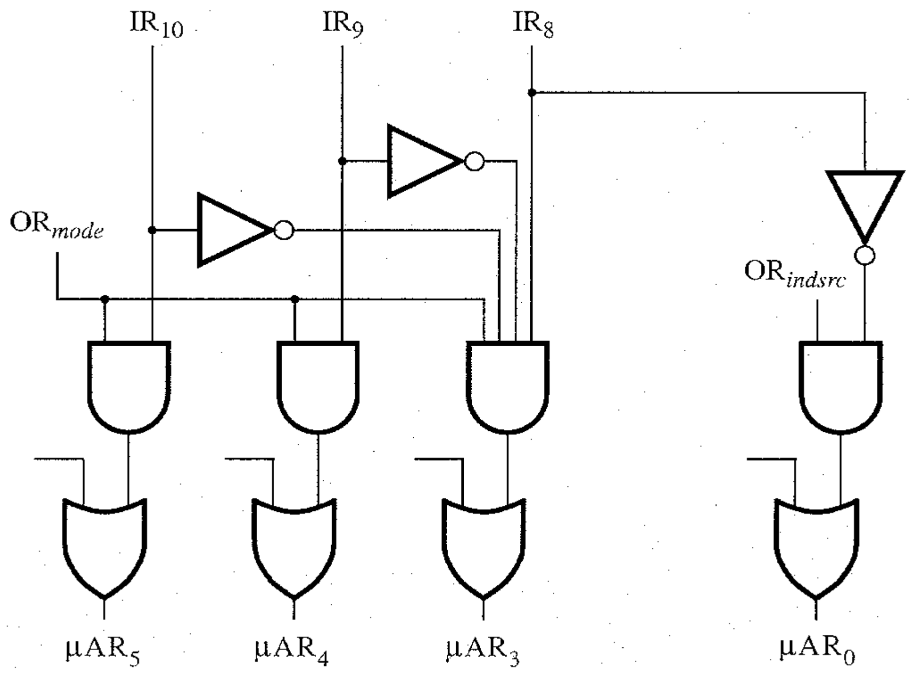
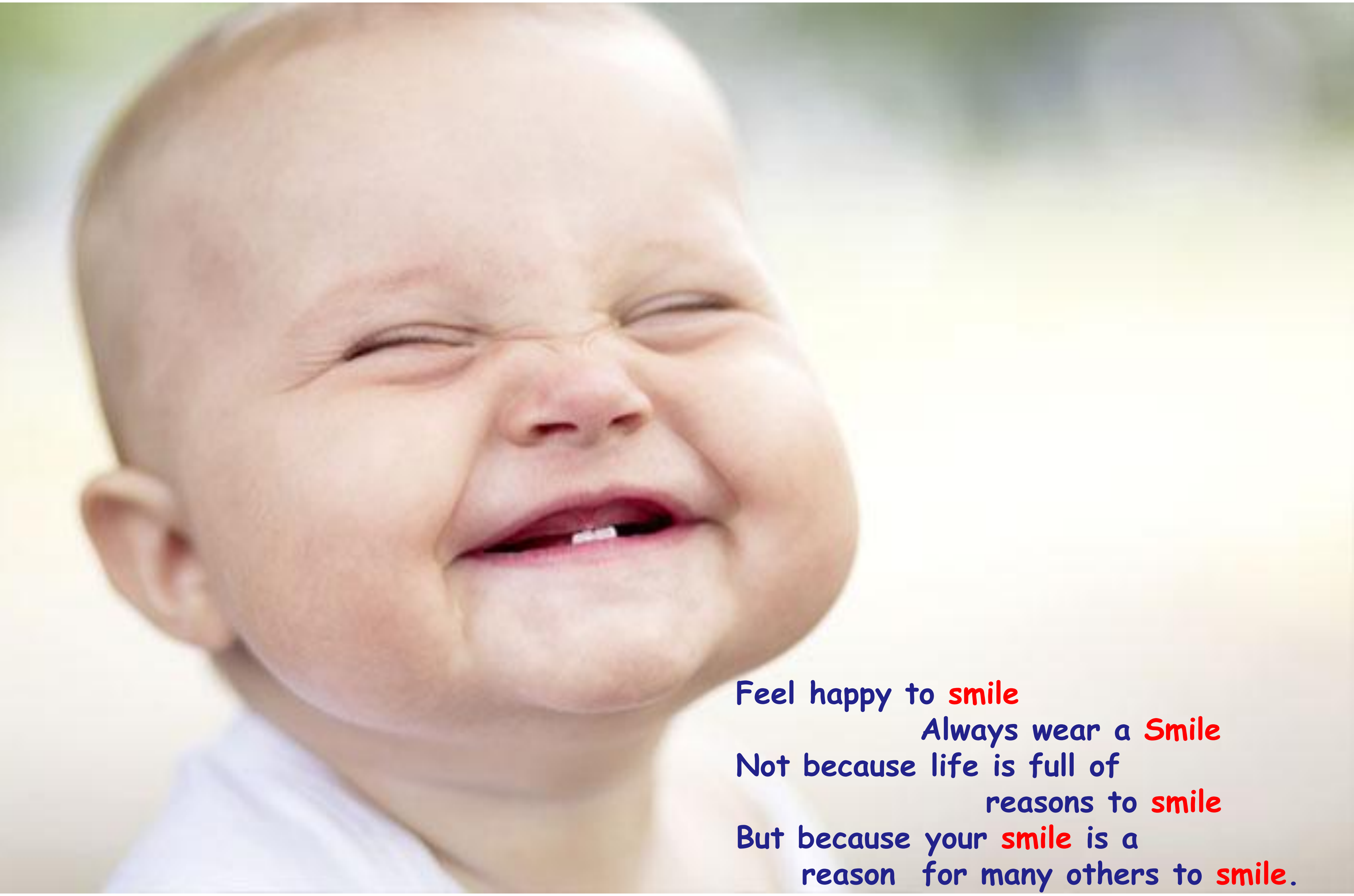


Figure 7.26. Control circuitry for bit-ORing

(part of the decoding circuits in Figure 7.25).



Feel happy to smile
Always wear a Smile
Not because life is full of
reasons to smile
But because your smile is a
reason for many others to smile.

Smile !
God loves you !
Keep smiling and others will too !

Don’t forget to smile

Always
remember to be happy
because
you never know
who's falling in love
with your smile.
fanz wave
If you don't have a smile,
I' give you one of mine.

All the best

Do well in the examinations

Wish you all success

God bless you.

THANK YOU